UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE CIÊNCIAS
DEPARTAMENTO DE COMPUTAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

Dissertação de Mestrado

Usando Assertivas de Correspondência para Especificação e Geração de Visões XML para Aplicações *Web*

FERNANDO CORDEIRO DE LEMOS

FORTALEZA/CE
2007

UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE CIÊNCIAS
DEPARTAMENTO DE COMPUTAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

Usando Assertivas de Correspondência para Especificação e Geração de Visões XML para Aplicações *Web*

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientadora: Profª. Drª. Vânia Maria Ponte Vidal

FERNANDO CORDEIRO DE LEMOS

UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE CIÊNCIAS
DEPARTAMENTO DE COMPUTAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

Usando Assertivas de Correspondência para Especificação e Geração de Visões XML para Aplicações *Web*

Fernando Cordeiro de Lemos

Orientadora: Profª. Drª. Vânia Maria Ponte Vidal

Aprovada em __/__/_____

BANCA EXAMINADORA

Profª. Drª. Vânia Maria Ponte Vidal (Orientadora)
Universidade Federal do Ceará - UFC

Prof. Dr. Marco Antônio Casanova
Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO

Prof.ª Dr.ª Bernadette Farias Lóscio
Universidade Federal do Ceará – UFC

AGRADECIMENTOS

A Deus, por tudo e principalmente por ter iluminado meu caminho durante essa oportunidade de crescimento e aprendizado.

A minha família, que sempre me apóia e acredita em mim.

A minha querida orientadora Vânia Vidal pela persistência, crença, incentivo, ensinamentos e dedicação muito bem empregados no acompanhamento e orientação desta dissertação.

Ao Professor Dr. Marco Antônio Casanova e à Professora Dr.ª Bernadette Farias Lóscio, por terem aceitado o convite para participar da banca examinadora e por colaborarem para a melhoria deste trabalho.

Aos meus queridos amigos Fábio, Marcel, Valdiana, Lineu, Wamberg, Berna, Eveline e todos que já passaram pela minha vida e me ajudaram com a sua amizade, idéias e também com conselhos.

A Universidade Federal do Ceará pela oportunidade do grande crescimento profissional e intelectual.

Aos professores e funcionários do Mestrado em Ciência da Computação da Universidade Federal do Ceará pelo apoio à realização do curso.

A Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo auxílio financeiro que possibilitou a realização deste trabalho.

ABSTRACT

Web applications that have large number of pages, whose contents are dynamically extracted from one or more databases, and that requires data intensive access and update, are known as "data-intensive Web applications" (DIWA applications) [7]. In this work, the requirements for the content of each page of the application are specified by an XML view, which is called Navigation View (NV). We believe that the data of NVs are stored in a relational or XML database.

In this work, we propose an approach to specify and generate NVs for Web applications whose content is extracted from one or more data sources. In the proposed approach, a NV is specified conceptually with the help of a set of Correspondence Assertions [44], so that the definition of NV can be generated automatically based on assertions of view.

RESUMO

Aplicações *Web* que possuem grande número de páginas, cujos conteúdos são dinamicamente extraídos de banco de dados, e que requerem intenso acesso e atualização dos dados, são conhecidas como "*data-intensive Web applications*" (aplicações DIWA) [7]. Neste trabalho, os requisitos de conteúdo de cada página da aplicação são especificados através de uma visão XML, a qual denominamos Visão de Navegação (VN). Consideramos que os dados das VNs estão armazenados em um banco de dados relacional ou XML.

Nesse trabalho, propomos um enfoque para especificação e geração de VNs para aplicações *Web* cujo conteúdo é extraído de uma ou mais fontes de dados. No enfoque proposto, uma VN é especificada conceitualmente com a ajuda de um conjunto de Assertivas de Correspondência [44], de forma que a definição da VN pode ser gerada automaticamente a partir das assertivas da visão.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1
Introdução

1.1 Aplicações *Web*

A *Web* é um dos meios mais utilizados para disseminação e compartilhamento de informações atualmente. Navegando em sites *Web*, usuários podem ter acesso a inúmeros documentos e informações disponibilizados. Esses sites são formados por um conjunto de páginas, onde é permitida a navegação dos usuários para a realização de uma tarefa. As páginas *Web* possuem uma estrutura estática de interface com o usuário e conteúdo, a qual pode ser um documento estático, mas na maioria das vezes é extraído dinamicamente através de consultas a uma ou mais fontes de dados. Esse conteúdo costuma mudar rapidamente, pode estar armazenado em diferentes bases de dados e pode assumir uma variedade de formatos, estruturados ou não-estruturados [3].

Os sites que oferecem ao usuário a possibilidade de interferir na lógica de negócios são chamados de aplicações *Web* [8]. As aplicações *Web* que requerem acesso dinâmico a grande quantidade de dados, cujas estruturas de armazenamento são geralmente complexas, são, por sua vez, denominadas de *data-intensive Web applications*, ou aplicações DIWA [7].

Aplicações DIWA, em geral, são caracterizadas por requererem intenso acesso e atualização dos dados armazenados em bases de dados e utilizam uma arquitetura em três camadas como mostrada na Figura 1.1. Essa arquitetura separa a lógica da aplicação do repositório de dados, introduzindo uma distinção adicional de responsabilidades no lado do servidor e, ao mesmo tempo, garantindo o mínimo de esforço do lado do cliente. Toda a lógica de negócios é desempenhada pelo servidor durante o atendimento às requisições do lado cliente. Na *Camada Servidora de Dados*, os dados são armazenados de forma persistente e organizada. A *Camada Servidora de Aplicações* é composta pelos módulos que desempenham a lógica de negócios da aplicação e extraem do repositório os dados requeridos pelas visões dos usuários. A *Camada Cliente* representa o conteúdo das páginas *Web* geradas pela camada servidora de aplicações.

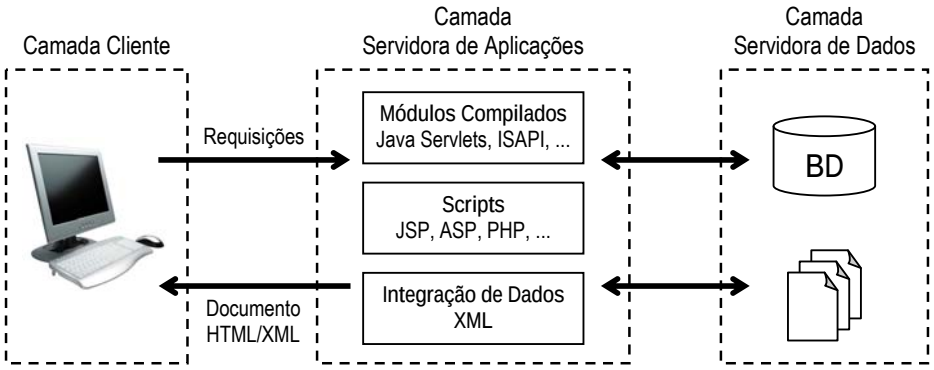

Figura 1.1. Arquitetura em três camadas

Existe atualmente uma grande variedade de tecnologias que são utilizadas para o desenvolvimento de aplicações *Web*. Essas tecnologias possibilitam a geração de conteúdo dinâmico e permitem os usuários da aplicação modificar sua lógica de negócio. As categorias de tecnologia mais utilizadas no desenvolvimento de uma aplicação *Web* são os módulos compilados e os *scripts* interpretados [8]. Algumas implementações mais populares dessas tecnologias são *Internet Server* API (ISAPI) [28] e *Java Servlets* [40], no caso de módulos compilados, e *Java Server Pages* (JSP) [41], *Active Server Pages* (ASP) [28] e PHP [32], no caso de *scripts*. Em uma aplicação *Web*, a escolha da tecnologia utilizada depende da natureza da aplicação, da organização e da própria equipe de desenvolvimento. Independentemente dessa escolha, a construção dessas aplicações necessita de um processo sistemático e bem definido que envolva as ferramentas e as metodologias desenvolvidas nas áreas de hipermídia, engenharia de software e projeto de banco de dados.

Diversas ferramentas estão sendo disponibilizadas no mercado para automatizar o desenvolvimento de aplicações *Web* utilizando as tecnologias citadas acima. Essas ferramentas trazem grandes benefícios aos programadores de aplicações, pois elas agilizam o desenvolvimento manual de páginas *Web*. Entretanto, um estudo cuidadoso sobre as características dessas ferramentas nos mostra que a maioria delas se concentra na implementação da aplicação, dispensando pouca ou nenhuma atenção ao processo geral de projetar aplicações *Web* [16].

1.2 Motivação

O desenvolvimento de aplicações *Web*, como de qualquer outro tipo de sistema, é um processo complexo, que deve resultar na satisfação de um grande número de usuários, com diferentes objetivos e que desejam realizar diferentes tarefas. Além disso, combinar uma navegação controlada pelo usuário com a facilidade de acesso aos dados e o grande volume de informações disponíveis na *Web* tornou necessário algumas adaptações no já conhecido processo incremental e interativo de desenvolvimento de sistemas utilizado na engenharia de software.

Do ponto de vista do usuário, a realização de uma tarefa envolve um determinado número de telas, onde cada tela contém uma estrutura estática de interface com o usuário (como menus e controles) e também conteúdo. Esse conteúdo pode ser um documento estático, mas muitas vezes é construído dinamicamente através de consultas a uma ou mais bases de dados. No nosso enfoque, os requisitos de conteúdo dinâmico de cada página de uma aplicação *Web* são especificados através de uma visão XML, denominada Visão de Navegação (VN). Cada interação entre o usuário e o sistema, necessária para realização de uma tarefa da aplicação, possui uma VN associada a ela, a qual especifica quais dados os usuários da aplicação manipulam no contexto dessa interação. Para cada tarefa da aplicação, utilizamos Diagramas de Interação dos Usuários (UID) [19][49] para representar graficamente as interações dos usuários com o sistema para a execução da tarefa.

Atualmente, a maioria das aplicações *Web* se enquadra na categoria de aplicações DIWA [7], uma vez que requerem intenso acesso e atualização dos dados armazenados em bases de dados. A especificação, construção e manutenção de visões de navegação não são tarefas fáceis para aplicações DIWA, pois, em geral, os conteúdos das VNs são gerados por módulos compilados, os quais representam uma grande quantidade de código *ad-hoc* específico da aplicação e da plataforma. Essa complexidade se traduz em um alto custo de desenvolvimento e manutenção. Manter essas visões também não é tarefa simples, uma vez que mudanças nos requisitos da aplicação ou no esquema da fonte de dados podem afetar uma grande quantidade de VNs que precisam, então, ser redefinidas. A especificação, construção e manutenção das VNs se torna ainda mais complexas quando os conteúdos das VNs são extraídos de múltiplas fontes de dados.

O problema de especificação, geração e manutenção das VNs não é tratado com rigor em nenhum dos métodos de desenvolvimento de aplicações *Web* conhecidos. Sem um método para especificação formal das VNs, não é possível automatizar a sua implementação e manutenção.

1.3 Trabalhos Relacionados

Diversas metodologias surgiram com o intuito de auxiliar os projetistas a superar as dificuldades no projeto de aplicações *Web* [36][37][3][16][18][7][14][15]. A maioria dessas metodologias apresenta arquitetura de esquemas como mostrado na Figura 1.2. Nessa arquitetura, cada VN é especificada sobre o Esquema Conceitual da Aplicação (ECA), o qual representa uma visão integrada dos requisitos de conteúdo da aplicação. O ECA, por sua vez, é especificado sobre os esquemas das fontes locais. O ECA compreende os requisitos de conteúdo da aplicação e define uma camada de independência lógica entre as VNs e esquemas das fontes locais. É importante dizer que a especificação conceitual de esquemas deve ser independente de sua apresentação, o que reduz drasticamente custos de desenvolvimento e manutenção.

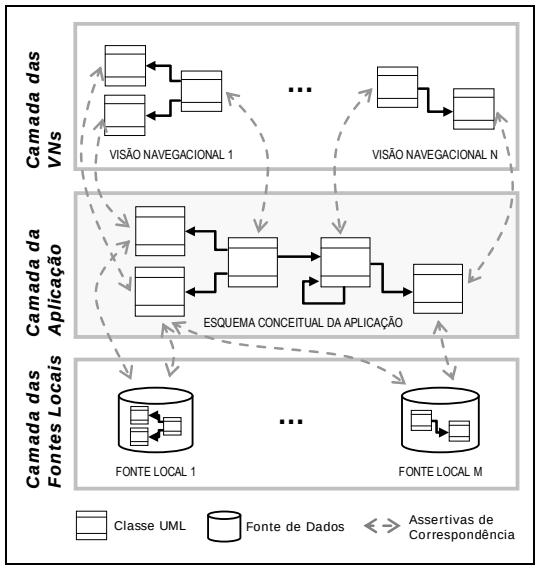

Figura 1.2. Arquitetura de esquemas

Grande parte dessas metodologias é baseada na utilização do padrão UML para desenvolvimento das aplicações. A UML fornece cenários e diagramas de caso de uso para levantamento e especificação das tarefas a serem realizadas pela aplicação. Além disso, os diagramas de classes da UML permitem a representação do domínio da aplicação. Para representar as interações entre usuários e o sistema, durante a realização

de uma tarefa descrita por um caso de uso, foram propostos os Diagramas de Interação dos Usuários (UIDs) [19][49].

O *Hypermedia Design Method* (HDM) [18] foi um dos métodos pioneiros para tratar do projeto de aplicações hipermídia e influenciou diversas metodologias subseqüentes, como OOHDM [35][36][37] e HDM-lite [16]. Todos esses métodos oferecem construtores que permitem o desenvolvimento de um rico projeto navegacional para as aplicações *Web*. Essas abordagens são baseadas em sistemas hipermídia, cujo objetivo é centrado na navegação de informações e apresentação.

A seguir, apresentamos algumas das principais metodologias existentes para o projeto de aplicações *Web*:

OOHDM

O OOHDM, *Object-Oriented Hypermedia Design Method*, é um método para desenvolvimento de aplicações *Web* que procura resolver as dificuldades encontradas na construção de projetos dessas aplicações, aplicando as estratégias da Engenharia de Software. A metodologia OOHDM apresenta uma abordagem orientada a objetos que utiliza *cenários* e *casos de uso* para especificação das tarefas a serem realizadas pela aplicação. *Diagramas de Interação de Usuários* (UIDs) [19][49] são utilizados para representação das interações entre os usuários e o sistema para a execução de cada tarefa. Em um UID, para cada interação é associada uma VN que especifica os requisitos de conteúdo no contexto dessa interação.

O modelo conceitual representa o domínio dos objetos da aplicação e seus relacionamentos. O esquema conceitual é construído a partir dos UIDs, como mostrado em [19][49], e é representado utilizando uma notação similar a UML.

A metodologia utiliza modelos navegacionais que permitem a construção de diferentes modelos a partir do mesmo modelo conceitual. Os modelos navegacionais são expressos por meio de dois esquemas: o Esquema de Classes Navegacionais e o Esquema de Contextos Navegacionais. No esquema de classes navegacionais, são definidos os objetos da aplicação que são navegáveis, onde as classes navegacionais representam visões dos objetos conceituais. No esquema de contextos navegacionais, a estrutura do espaço navegacional é representada em contextos. Cada contexto é definido pelas propriedades dos seus elementos, que pode ser baseado em seus atributos, em seus relacionamentos ou em ambos.

A separação entre os modelos conceitual e navegacional é bem definida: enquanto o modelo conceitual reflete os objetos e seus comportamentos no domínio da aplicação, o modelo navegacional trata da organização do hiper-espaço, levando em conta as tarefas e *profiles* dos usuários.

Na implementação, os objetos conceituais são armazenados de forma persistente (geralmente, em bancos de dados) e os objetos navegacionais e de interface são implementados como páginas *Web*. As interações dos UIDs são reexaminadas para determinar a navegação da aplicação final. Observa-se que, até este momento, todos os modelos foram construídos de forma a serem independentes da plataforma de implementação.

Em [25] é apresentado um método que estende o OOHDM, denominado SHDM (*Semantic Hypermedia Design Method*). O SHDM utiliza os conceitos de ontologia para enriquecer o modelo conceitual e navegacional da aplicação, trazendo as aplicações desenvolvidas para o contexto da *Web* Semântica [5].

Weave

A metodologia Weave [14][15] tem como base a especificação declarativa do site através de um modelo lógico, o qual é derivado de um modelo de grafo para dados XML. O modelo lógico descreve as classes da aplicação e seus relacionamentos. Cada classe modela uma coleção de páginas homogêneas, ou seja, páginas que representam uma mesma entidade na aplicação. Desta forma, cada página pode ser vista como uma instância de uma classe particular, identificada por um conjunto de parâmetros. A especificação de cada página inclui: (i) a declaração dos parâmetros que identificam uma instância da classe; (ii) a consulta SQL que retorna todas as possíveis instâncias da classe para os parâmetros especificados anteriormente; (iii) uma consulta que determina como os dados obtidos do banco são mapeados em elementos da página; e (iv) os *hyperlinks* que partem das instâncias da classe. As classes são especificadas através da linguagem *WeaveL*.

A materialização dos dados propriamente dita é baseada num sistema de *caching* de dados. A linguagem *WeaveL* neste momento é usada também para especificar as ações de materialização da classe.

Hera

A metodologia Hera [43], ao contrário das metodologias citadas anteriormente, aborda aplicações *Web* de integração de dados. A metodologia utiliza diferentes modelos (modelo de integração, modelo da aplicação, modelo de adaptação) para representar os diferentes aspectos envolvidos no desenvolvimento de aplicações *Web*. Os modelos de integração e de adaptação permitem a geração de diferentes apresentações da mesma aplicação.

A linguagem RDF (*Resource Description Framework*) [50] é utilizada para modelar os diferentes esquemas usados nas atividades da metodologia. As fontes locais devem prover seus dados em RDF e fornecer interface de consulta em RQL (*RDF Query Language*) [22]. O modelo da aplicação (esquema conceitual) é especificado em termos dos esquemas das fontes através de uma linguagem declarativa.

A metodologia utiliza um mediador próprio para a integração dos dados. O mediador recebe uma consulta RQL estendida, a qual contém informações sobre as fontes a serem consultadas e que dados obter. Em seguida essa consulta estendida é quebrada em consultas para as fontes locais. Os dados são então integrados e compatibilizados com o esquema conceitual no mediador. A compatibilização é necessária uma vez que a linguagem RQL não permite reestruturação dos dados.

Algumas das metodologias apresentadas [3][7][16] surgiram com o intuito de apresentar um processo sistemático para desenvolvimento de aplicações *Web*. Além do projeto conceitual e navegacional da aplicação, essas metodologias se preocupam também com a geração automática das páginas, de forma a minimizar os custos com a manutenção da aplicação.

Apesar de as metodologias apresentadas usarem especificações declarativas, tais especificações são baseadas em consultas SQL, as quais são difíceis de gerar e manter. Além disso, nenhuma das metodologias mencionadas lida realmente com o problema da especificação, geração e manutenção das Visões de Navegação, e sem uma especificação formal não é possível automatizar implementação e manutenção. Vale ressaltar que em aplicações *Web* mais complexas, as consultas que geram o conteúdo das páginas da aplicação são geralmente complexas, pois devem mapear os objetos do banco de dados em objetos de tipos complexos, onde os tipos dos objetos do banco de dados e da VN podem ter estruturas bem diferentes.

1.4 Contribuições da Dissertação

O objetivo da dissertação é um processo para especificação e geração de VNs para aplicações *Web* cujo conteúdo é extraído de uma ou mais fontes de dados. Consideramos neste trabalho fontes de dados relacionais e XML, por serem as mais comuns. As visões de navegação são implementadas como visões XML, as quais são especificadas conceitualmente usando o formalismo das assertivas de correspondência [44] proposto neste trabalho. O formalismo proposto para a especificação das VNs é independente de linguagem de consulta e permite que as definições das VNs sejam geradas automaticamente.

As principais contribuições são:

1. Proposta de um formalismo para especificação e geração de visões XML sobre uma base de dados relacional. No formalismo proposto, utilizamos assertivas de correspondência para definir o mapeamento entre a visão XML e o esquema relacional. Definimos formalmente as condições sob as quais um conjunto de assertivas de correspondência especifica completamente uma visão XML em termos do esquema base e, no caso de base relacional, mostramos que os mapeamentos definidos pelas assertivas da visão podem ser expressos como uma consulta SQL/XML. Propomos um algoritmo que, baseado nas assertivas de correspondência da visão, gera a consulta SQL/XML.
2. Proposta de um formalismo para especificar visões XML sobre múltiplas fontes de dados, denominadas Visões de Integração de Dados. Tal formalismo é uma extensão do formalismo usado para especificar visões XML sobre uma única base. Além disso, propomos um algoritmo para gerar, a partir das assertivas de uma visão de integração de dados, o plano de materialização da visão, o qual computa o estado da visão a partir dos estados das fontes locais.
3. Proposta de um processo para especificação e geração das VNs para aplicações *Web* com base nos formalismos citados no itens 1 e 2. O processo proposto consiste de duas fases: *Projeto Conceitual* e *Projeto Lógico*. No *Projeto Conceitual* são obtidos os esquemas conceituais das VNs, o Esquema Conceitual da Aplicação e os mapeamentos entre as VNs e o ECA e entre o ECA e o esquema da fonte local (ou esquemas das fontes locais). No *Projeto Navegacional*, as consultas ou plano de materialização que extraem o conteúdo das VNs são gerados automaticamente a partir de suas assertivas de correspondência (mapeamentos).

1.5 Organização da Dissertação

Os capítulos a seguir estão organizados da seguinte forma. No Capítulo 2, apresentamos uma abordagem para a especificação e geração de visões XML sobre dados relacionais em SGBDs que suportem visões SQL/XML. No Capítulo 3, apresentamos um formalismo para especificação de visões XML virtuais sobre múltiplas fontes de dados (Visões de Integração de Dados). No Capítulo 4, apresentamos um processo para a especificação e geração de visões de navegação para aplicações *Web* de intenso acesso e manipulação de dados. No Capítulo 5, são apresentadas as conclusões e sugestões para trabalhos futuros.

CAPÍTULO 2
Especificação e Geração de Visões XML sobre Base de Dados Relacional

Neste capítulo, apresentamos uma abordagem para a especificação e geração de visões XML sobre dados relacionais em SGBDs que possuam mecanismo de visão SQL/XML. O capítulo está organizado da seguinte forma: A Seção 2.1 apresenta uma introdução sobre visões XML; A Seção 2.2 discute coleções XML e algumas definições básicas; A Seção 2.3 discute detalhes do padrão SQL/XML; A Seção 2.4 discute visões XML; A Seção 2.5 apresenta nosso formalismo para especificar visões XML sobre uma base de dados relacional; A Seção 2.6 discute como especificar visões SQL/XML usando assertivas de correspondência [44]; A Seção 2.7 apresenta o algoritmo que gera automaticamente a definição da visão SQL/XML a partir das assertivas de correspondência; A Seção 2.8 apresenta a ferramenta XVBA, que auxilia no processo de especificação, geração e manutenção de visões XML sobre base relacional; A Seção 2.9 apresenta as conclusões.

2.1 Introdução

O problema de publicar dados relacionais no formato de XML tem especial importância, uma vez que XML emergiu como o formato padrão da troca de dados entre aplicações *Web* enquanto a maioria dos dados de negócio continua armazenada em SGBDs relacionais. Uma maneira geral e flexível de publicar dados relacionais no formato XML é através da criação de *Visões XML* sobre dados relacionais [38][48].

Uma visão XML pode ser *virtual* ou *materializada*. Visões materializadas melhoram o desempenho das consultas e a disponibilidade de dados, mas devem ser atualizadas para refletir mudanças nos dados da fonte subjacente [44]. No caso de visões virtuais, os dados ainda persistem em bases relacionais, enquanto as aplicações acessam os dados no formato de XML através da visão [1]. Entretanto, criar visões XML de dados relacionais levanta o problema de como definir a visão e de como avaliar uma consulta XQuery [52] sobre a visão. Neste caso, a consulta XQuery deve ser traduzida

em SQL através da composição desta com a definição da visão. Neste trabalho, tratamos somente de visões XML virtuais.

A publicação de dados relacionais através de visões XML foi tratada, por exemplo, em *XPeranto* [6][21][39] e em *SilkRoute* [12][13]. Em ambos os trabalhos, a visão XML é definida como uma consulta XQuery sobre uma visão XML canônica que representa as tabelas da base de dados e seus atributos. Esta consulta especifica o esquema da visão e os mapeamentos, os quais descrevem como o esquema da visão está relacionado com a visão canônica. A avaliação de uma consulta XQuery sobre a visão é executada usando um *middleware* no topo da base de dados relacional. O *middleware* traduz a consulta em consultas SQL equivalentes. Em seguida, os resultados SQL são etiquetados (*tagged*) para produzir o documento XML resultante. Nestes sistemas, não há garantia de processamento eficiente de consultas.

No *DB2 XMLExtender* [4] e no *SQL Server* [34], os mapeamentos são armazenados dentro de *esquemas anotados* (*annotated schemas*) [34]. Em ambos os casos, a definição dos mapeamentos é muito complexa. Além disso, o SQL Server provê a cláusula FOR XML para a transformação de resultados de consultas em XML. Neste caso, o mapeamento é definido no tempo de execução da consulta e não é armazenado de maneira alguma, o que viola a premissa de transparência do mapeamento.

Com a introdução do tipo de dados XML [1] e do padrão SQL/XML [9] como parte do SQL:2003 [9], usuários podem recorrer às funções de publicação do SQL/XML para criar visões XML virtuais sobre esquemas relacionais. O Oracle [1] foi o primeiro SGBD a suportar, com seu módulo XMLDB [1], a criação de visões XML como consultas SQL/XML sobre os dados relacionais. As vantagens desta abordagem estão no uso de um padrão para publicar dados relacionais e na capacidade de processar as funções de publicação do SQL/XML juntamente com as cláusulas SQL, o que representa um ganho de desempenho [27]. Assim, a reescrita da consulta XQuery pode ser executada dentro do SGBD [23][30], ao contrário das soluções que utilizam *middlewares*. No entanto, apesar de muitos SGBDs implementarem o padrão SQL/XML, até o momento somente o Oracle provê um mecanismo de visão SQL/XML.

Entretanto, criar a definição de uma visão SQL/XML exige conhecimento avançado da linguagem e é uma tarefa que consome tempo. Além disso, os usuários terão que redefinir a visão sempre que o esquema relacional base mudar.

Neste capítulo, apresentamos uma abordagem para a especificação e criação de visões SQL/XML sobre dados relacionais. A definição da visão SQL/XML é obtida das assertivas de correspondência [44] da visão, as quais especificam formalmente [11] o mapeamento entre o esquema da visão e o esquema relacional. No caso de visões materializadas, as regras necessárias para manter a visão podem ser geradas automaticamente com base nas assertivas de correspondência da visão, como mostrado em [44].

Este capítulo apresenta três contribuições principais:

- Primeiramente, propomos o uso das assertivas de correspondência [33][44] para especificar o mapeamento entre o esquema da visão XML e o esquema relacional. Nós especificamos formalmente as circunstâncias sob as quais um conjunto de assertivas de correspondência especifica completamente a visão XML em termos do esquema relacional. Por fim, mostramos que os mapeamentos definidos pelas assertivas de correspondência da visão podem ser expressos como uma consulta SQL/XML, a qual define a visão.
- Em segundo lugar, propomos um algoritmo que, baseado nas assertivas de correspondência da visão, gera a consulta SQL/XML que constrói os elementos da visão XML a partir das tuplas relacionais.
- Em terceiro lugar, propomos a ferramenta *XML View-By-Assertions* (XVBA) que facilita a tarefa de criação e manutenção de visões SQL/XML. Os formalismos usados por outras ferramentas de mapeamento ou são ambíguos [20] ou requerem que o usuário declare mapeamentos lógicos complexos [53][17].

2.2 Tipos e Coleções XML

Nesta seção apresentamos conceitos de tipos de coleções XML, os quais serão utilizados no Capítulo 3.

Tipo XML

Em XML, uma definição de tipo é usada para descrever o conteúdo de um elemento, sua lista de atributos e as restrições sobre as suas propriedades. As ***propriedades de um tipo*** definem seus elementos e atributos. Um tipo XML pode ser *simples* ou *complexo*. Um tipo simples restringe o conteúdo de um atributo, ou o conteúdo textual de um elemento. Um tipo complexo restringe o conteúdo permitido para elementos, em termos dos seus atributos ou sub-elementos.

```
1.  <complexType name="TLivro ">
2.    <sequence>
3.      <element name="ISBN" type="string"/>
4.      <element name="titulo" type="string"/>
5.      <element name="resumo" type="string"/>
6.      <element name="ano" type="gYear"/>
7.      <element name="autor" type="TAutor "
8.                maxOccurs="unbounded"/>
9.    </sequence>
10. </complexType>
11. <complexType name="TAutor ">
12.   <sequence>
13.     <element name="email" type="string"/>
14.     <element name="nome" type="string"/>
15.   </sequence>
16. </complexType>
```

Figura 2.1. Declaração de TLivro

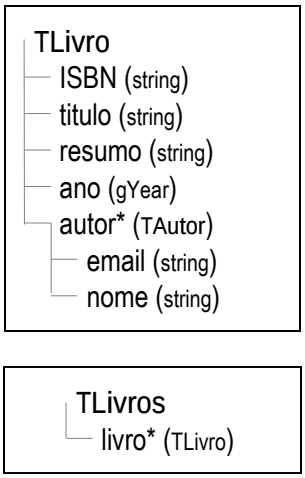

Figura 2.2. Representações gráficas de TLivro e TLivros

Um tipo complexo T é *restrito* sse T é definido usando somente os construtores *complexType* e *sequence* do XML Schema, e o tipo de seus atributos é um tipo XML simples. Os tipos simples não precisam ser definidos, pois a especificação do XML Schema contempla um conjunto expressivo de tipos primitivos [51]. Neste trabalho, consideramos somente tipos complexos restritos.

Por exemplo, mostra a declaração dos tipos complexos restritos TLivro e TAutor.

Utilizamos uma representação estruturada em "árvore de diretórios" para representar um tipo complexo, onde cada nó da árvore representa um elemento ou atributo declarado no respectivo tipo. Para cada declaração de elemento ou atributo é gerado um nó na árvore. Se o elemento for de um tipo complexo, será gerado um nó-filho na árvore para cada item da sua estrutura. Cada nó deve ser rotulado com o nome do elemento ou atributo, seu tipo e sua restrição de ocorrência. Para atributos, seu nome é precedido do símbolo "@". Se o elemento for de ocorrência múltipla, a restrição de ocorrência é "*".

 mostra a representação gráfica do tipo TLivro. O primeiro nó da árvore é o nome do tipo. Os nós filhos de TLivro são as suas propriedades: ISBN, titulo, resumo, ano e autor. Como autor é um elemento de tipo complexo TAutor, é gerado um nó-filho para cada propriedade de TAutor: email e nome.

Definimos *tipo coleção* como sendo um tipo restrito composto por uma única definição de elemento multiocorrência. Por exemplo, o tipo TLivros () é um tipo coleção.

Coleção XML

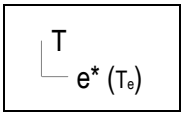

Figura 2.3. Representação gráfica do tipo coleção T

Uma *Coleção XML* é uma instância de um tipo coleção. Suponha C uma coleção XML instância do tipo T mostrado na Figura 2.3. O esquema de C é uma tupla <e, T_e>, onde:

(i) e é o nome dos elementos primários da coleção C;

(ii) T_e é o tipo de e.

Um *estado da coleção* C é um conjunto não-ordenado de elementos XML do tipo T_e. Para facilitar a notação, denotamos por $C o estado de uma coleção XML C.

Considere, por exemplo, a coleção XML Livros cujo esquema é <livro, TLivro>. Figura 2.4 mostra um exemplo de estado da coleção Livros, o qual consiste de um conjunto de elementos <*livro*> de tipo TLivro.

Uma importante restrição nos elementos de uma coleção é a restrição de chave. Uma *chave de uma coleção* é um conjunto de propriedades de tipo simples cujos valores identificam um único elemento da coleção. {ISBN} é um exemplo de chave para a coleção Livros.

```
<livro>
 <ISBN>1558605622</ISBN>
 <título>Understanding SQL and Java Together</título>
 <resumo>Database vendors like Oracle and Sybase have
signed on quickly with Java support...</resumo>
 <ano>2000</ano>
 <autor>
   <email>andrew.eisenberg@us.ibm.com</email>
   <nome>Andrew Eisenberg</nome>
 </autor>
 <autor>
   <email>jim.melton@acm.org</email>
   <nome>Jim Melton</nome>
 </autor>
</livro>
<livro>
 <ISBN>1558607110</ISBN>
 <título>Querying XML</título>
 <resumo>XML has become the lingua franca for...</resumo>
 <ano>2006</ano>
 <autor>
   <email>jim.melton@acm.org</email>
   <nome>Jim Melton</nome>
 </autor>
 <autor>
   <email>stephen.buxton@oracle.com</email>
   <nome>Stephen Buxton</nome>
 </autor>
</livro>
<livro>
 <ISBN>1558608435</ISBN>
 <título>Designing Data-Intensive Web Applications</título>
 <resumo>As described by the authors, the goal of this book
is the proposal...</resumo>
 <ano>2002</ano>
 <autor>
   <email>ceri@elet.polimi.it</email>
   <nome>Stefano Ceri</nome>
 </autor>
 <autor>
   <email>brambilla@elet.polimi.it</email>
   <nome>Marco Brambilla</nome>
 </autor>
 <autor>
   <email>fraterna@elet.polimi.it</email>
   <nome>Piero Fraternali</nome>
 </autor>
 <autor>
   <email>matera@elet.polimi.it</email>
   <nome>Maristella Matera</nome>
 </autor>
 <autor>
   <email>comai@elet.polimi.it</email>
   <nome>Sara Comai</nome>
 </autor>
</livro>
```

Figura 2.4. Exemplo de um estado da coleção Livros

Expressão de Caminho

Uma expressão de caminho localiza um nó em uma árvore e retorna uma seqüência de nós distintos na ordem do documento. Tal expressão consiste em um ou mais passos. Cada passo representa um movimento ao longo do documento, em uma determinada direção, e retorna uma lista de nós que servem como um ponto de partida para o próximo passo.

Sejam $T_1,\ldots,T_n$ tipos XML. Suponha que T_k possui uma propriedade p_k de tipo T_{k+1}, para k=1,...,n-1. Dizemos que:

(i) $\delta = p_1 / p_2 /.../ p_{n-1}$ é um *caminho* de T_1;

(ii) T_n é o tipo de δ; e

(iii) δ *tem ocorrência simples* sse p_k tem ocorrência simples, para k=1,...,n-1; caso contrário, δ *tem ocorrência múltipla*.

Por exemplo, dado um elemento $t da coleção Livros, a expressão de caminho $t/autor/nome retorna os elementos <*nome*> que são filhos dos elementos <*autor*> que são filhos de $t.

Coleções Aninhadas

Considere C uma coleção XML de esquema <e,T> e $\delta = e_1/.../e_n$ um caminho multiocorrência de T. C(δ) denota uma *coleção aninhada de* C, de esquema $\langle e_n, T_{e_n}\rangle$, e cujo estado é definido por:

$$\{ \$p \mid \exists \$c \in \$C \text{ tal que } \$p \in \$c/\delta \}.$$

Considere o estado da coleção livros na Figura 2.4. Livros(autor) é uma coleção aninhada de Livros, a qual tem esquema <autor, TAutor>, e cujo estado é mostrado na Figura 2.5.

```
<autor>
  <email>andrew.eisenberg@us.ibm.com</email>
  <nome>Andrew Eisenberg</nome>
</autor>
<autor>
  <email>stephen.buxton@oracle.com</email>
  <nome>Stephen Buxton</nome>
</autor>
<autor>
  <email>brambilla@elet.polimi.it</email>
  <nome>Marco Brambilla</nome>
</autor>
<autor>
  <email>jim.melton@acm.org</email>
  <nome>Jim Melton</nome>
</autor>
<autor>
  <email>ceri@elet.polimi.it</email>
  <nome>Stefano Ceri</nome>
</autor>
<autor>
  <email>jim.melton@acm.org</email>
  <nome>Jim Melton</nome>
</autor>
<autor>
  <email>fraterna@elet.polimi.it</email>
  <nome>Piero Fraternali</nome>
</autor>
<autor>
  <email>matera@elet.polimi.it</email>
  <nome>Maristella Matera</nome>
</autor>
<autor>
  <email>comai@elet.polimi.it</email>
  <nome>Sara Comai</nome>
</autor>
```

Figura 2.5. Estado da coleção aninhada Livros(autor) correspondente ao estado da coleção **Livros** na Figura 2.4

Ligações entre Coleções

Sejam

- C_1 e C_2 coleções (globais ou aninhadas) de esquemas $<e_1,T_1>$ e $<e_2,T_2>$, respectivamente;
- a_1, ..., a_n propriedades de T_1; e
- b_1, ..., b_n propriedades de T_2.

A *função de mapeamento* f:$T_1\{a_1, ..., a_n\} \rightarrow T_2\{b_1, ..., b_n\}$, mapeia uma instância de T_1 em instâncias de T_2, tal que dada uma instância $\$t_1$ do tipo T_1, temos que

$$f(\$t_1) = \{ \$t_2 \mid \$t_2/bi = \$t_1/a_i, 1 \leq i \leq n\}.$$

No caso em que $\{a_1, ..., a_n\}$ é uma chave de C_1 e $\{b_1, ..., b_n\}$ é uma chave para C_2, dizemos que existe uma *ligação de* C_1 *para* C_2 *através de f*, denotada por ℓ:$C_1 \xrightarrow[f]{} C_2$, onde ℓ é o nome da ligação. Neste caso f define um mapeamento 1-1 dos elementos de C_1 em elementos de C_2.

Dado um elemento $\$c_1$ de $\$C_1$, a expressão de caminho $\$c_1/\ell$ retorna o elemento $\$c_2$ de C_2 tal que $\$c_2$ é referenciado pelo elemento $\$c_1$ através de f. Mais formalmente,

$$\$c_1/\ell = \{ \$c_2 \in \$C_2 \mid f(\$c_1)=\$c_2\}.$$

Considere, por exemplo, a coleção Autores, a qual contém um conjunto de elementos <*autor*> do tipo TAutor' (Figura 2.6). Suponha que o estado de Autores é aquele mostrado na Figura 2.7. Considere também a coleção aninhada Livros(autor) da coleção Livros. Suponha a função de mapeamento f:TAutor{email}$\rightarrow$TAutor'{email}, onde {email} é uma chave da coleção Autores. Assim, ℓ:Livros(autor) $\xrightarrow[f]{}$ Autores é uma ligação da coleção aninhada Livros(autor).

Considere o elemento $l em Livros tal que $l = $Livros[ISBN = 1558608435]. A expressão $l/autor/$\ell$ retorna a coleção XML mostrada na Figura 2.8. Note que podem existir autores na coleção Livros(autor) que não têm correspondente em Autores. É o caso do elemento Livros/autor[email=stephen.buxton@oracle.com], o qual não tem correspondente em $l/autor/$\ell$.

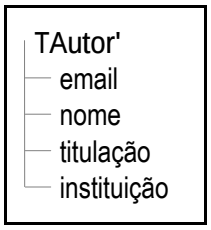

Figura 2.6. Tipo XML TAutor'

```
<autor>
 <email>fraterna@elet.polimi.it</email>
 <nome>Piero Fraternali</nome>
 <titulação>Ph.D.</titulação>
 <instituição>Politecnico di Milano</instituição>
</autor>
<autor>
 <email>andrew.eisenberg@us.ibm.com</email>
 <nome>Andrew Eisenberg</nome>
 <titulação>Ph.D.</titulação>
 <instituição>IBM</instituição>
</autor>
<autor>
 <email>jim.melton@acm.org</email>
 <nome>Jim Melton</nome>
 <titulação>Ph.D.</titulação>
 <instituição>Oracle</instituição>
</autor>
<autor>
 <email>comai@elet.polimi.it</email>
 <nome>Sara Comai</nome>
 <titulação>Ph.D.</titulação>
 <instituição>Politecnico di Milano</instituição>
</autor>
<autor>
 <email>brambilla@elet.polimi.it</email>
 <nome>Marco Brambilla</nome>
 <titulação>Ph.D.</titulação>
 <instituição>Politecnico di Milano</instituição>
</autor>
<autor>
 <email>ceri@elet.polimi.it</email>
 <nome>Stefano Ceri</nome>
 <titulação>Ph.D.</titulação>
 <instituição>Politecnico di Milano</instituição>
</autor>
<autor>
 <email>matera@elet.polimi.it</email>
 <nome>Maristella Matera</nome>
 <titulação>Ph.D.</titulação>
 <instituição>Politecnico di Milano</instituição>
</autor>
<autor>
 <email>ioana.manolescu@inria.fr</email>
 <nome>Ioana Manolescu</nome>
 <titulação>Ph.D.</titulação>
 <instituição>INRIA</instituição>
</autor>
```

Figura 2.7. Estado da coleção Autores

```
<autor>
 <email>fraterna@elet.polimi.it</email>
 <nome>Piero Fraternali</nome>
 <titulação>Ph.D.</titulação>
 <instituição>Politecnico di Milano</instituição>
</autor>
<autor>
 <email>comai@elet.polimi.it</email>
 <nome>Sara Comai</nome>
 <titulação>Ph.D.</titulação>
 <instituição>Politecnico di Milano</instituição>
</autor>
<autor>
 <email>brambilla@elet.polimi.it</email>
 <nome>Marco Brambilla</nome>
 <titulação>Ph.D.</titulação>
 <instituição>Politecnico di Milano</instituição>
</autor>
<autor>
 <email>ceri@elet.polimi.it</email>
 <nome>Stefano Ceri</nome>
 <titulação>Ph.D.</titulação>
 <instituição>Politecnico di Milano</instituição>
</autor>
<autor>
 <email>matera@elet.polimi.it</email>
 <nome>Maristella Matera</nome>
 <titulação>Ph.D.</titulação>
 <instituição>Politecnico di Milano</instituição>
</autor>
```

Figura 2.8. Resultado da expressão Livros[ISBN=1558608435]/autor/ℓ

2.3 O Padrão SQL/XML

Grande parte dos SGBDs mais importantes possui extensões proprietárias para a geração de XML a partir de dados relacionais. Tais extensões usam diferentes abordagens e não existe interoperabilidade entre elas, obrigando usuários que trabalham com diversos bancos a escreverem códigos diferentes, um para cada SGBD.

SQL/XML é um padrão ANSI/ISO desenvolvido pelo INCITS H2.3 que estende o SQL. Com SQL/XML é possível construir consultas declarativas que retornam dados XML a partir de bases relacionais. O XML resultante da consulta pode ter a estrutura que o projetista desejar, e as consultas podem ser arbitrariamente complexas. Para um programador SQL, SQL/XML é fácil de aprender, pois envolve um grupo pequeno de adições à já existente linguagem SQL. Uma vez que SQL é uma linguagem madura, existe um grande número de ferramentas e infra-estrutura disponíveis que podem ser facilmente adaptadas para suportar o novo padrão.

No padrão SQL/XML, as *funções de publicação* constituem a parte que provê a publicação de dados relacionais no formato XML. As funções de publicação são usadas diretamente na consulta SQL e permitem criar qualquer estrutura XML que o usuário deseje a partir do esquema relacional. O resultado de uma função de publicação é sempre uma instância do tipo de dados XML, o qual também é definido pelo padrão. Em uma consulta SQL/XML, as funções de publicação podem ser processadas pelo SGBD juntamente com as cláusulas SQL, o que representa um ganho de performance [27].

As principais funções de publicação do SQL/XML são: *XMLElement()*, *XMLAttributes()*, *XMLForest()*, *XMLAgg()*, descritas a seguir:

XMLElement e XMLAttributes – A função *XMLElement* constrói um novo elemento XML com atributos e conteúdo. A função recebe como parâmetro o nome do elemento, um conjunto opcional de atributos para o elemento, e zero ou mais argumentos adicionais que compõem o conteúdo do elemento. O resultado da função é uma instância do tipo XMLType. Caso todos os argumentos da função retornem valor nulo, então nenhum conteúdo é gerado para o elemento.

A função *XMLAttributes* especifica os atributos do elemento raiz. A função recebe como argumento um ou mais expressões escalares que podem apresentar *aliases* ou não. Se o valor de alguma expressão for nulo, o atributo correspondente não é gerado. A função XMLAttributes só pode ser chamada como primeiro argumento da função XMLElement.

Como exemplo, considere a seguinte consulta sobre a relação CLIENTES_REL:

```
SELECT XMLElement("cliente",
          XMLAttributes(C.CCODIGO AS "codigo"),
          XMLElement("nome", C.CNOME),
          XMLElement("fone", C.CFONE1),
          XMLElement("fone", C.CFONE2),
          XMLElement("fone", C.CFONE3) )
FROM CLIENTES_REL C;
```

O resultado da consulta é dado por:

```
<cliente codigo="193">
   <nome>Bryan Huston</nome>
   <fone>+91 11 012 4813</fone>
   <fone>+91 11 083 4813</fone>
   <fone>+91 11 012 4827</fone>
</cliente
<cliente codigo="195">
   <nome>Cary Stockwell</nome>
   <fone>+91 11 012 4835</fone>
   <fone></fone>
   <fone></fone>
</cliente
```

O identificador cliente dá nome ao elemento raiz. A expressão '`C.CCODIGO AS "codigo"`' dentro da função XMLAttributes gera o atributo codigo do elemento a partir do valor da expressão escalar '`C.CCODIGO`'. As expressões '`XMLElement("nome", C.CNOME)`' e '`XMLElement("fone", C.CFONE1)`' geram o conteúdo do elemento <*cliente*>. Em '`XMLElement("nome", C.CNOME)`', o valor escalar da expressão '`C.CNOME`' é mapeado no valor XML equivalente [10], gerando assim o conteúdo do elemento <*nome*>. Note que, no segundo elemento <*cliente*> do resultado, como os valores de '`C.CFONE2`' e '`C.CFONE3`' são nulos, então nenhum conteúdo é gerado para o elemento correspondente.

XMLForest – A função *XMLForest* gera uma floresta de elementos XML a partir de seus argumentos, os quais são expressões escalares com *aliases* opcionais. Cada expressão é convertida no formato XML correspondente e, caso um *alias* tenha sido atribuído, este será o identificador do elemento gerado. Expressões que resultam em nulo não geram elementos.

Como exemplo, considere a seguinte consulta sobre a relação CLIENTES_REL:

```
SELECT XMLElement("cliente",
          XMLAttributes(C.CCODIGO AS "codigo"),
          XMLElement("nome", C.CNOME),
          XMLForest(C.CFONE1 AS "fone",
                    C.CFONE2 AS "fone",
                    C.CFONE2 AS "fone") )
FROM CLIENTES_REL C;
```

O resultado da consulta é dado por:

```
<cliente codigo="193">
   <nome>Bryan Huston</nome>
   <fone>+91 11 012 4813</fone>
   <fone>+91 11 083 4813</fone>
   <fone>+91 11 012 4827</fone>
</cliente
<cliente codigo="195">
   <nome>Cary Stockwell</nome>
   <fone>+91 11 012 4835</fone>
</cliente
```

O identificador fone dá nome aos elementos gerados das expressões 'C.CFONE1 AS "fone"', 'C.CFONE2 AS "fone"' e 'C.CFONE3 AS "fone"'. Note que, no segundo elemento <cliente> do resultado, como os valores de 'C.CFONE2' e 'C.CFONE3' são nulos, então nenhum elemento foi gerado.

XMLAgg – É uma função de agregação que gera uma agregado XML (uma lista de elementos XML) a partir de cada agrupamento SQL. Se nenhum agrupamento for especificado, é retornado um agregado XML para todas as cláusulas da consulta. Argumentos nulos são removidos do resultado. A função recebe como parâmetro uma única expressão que gera uma instância do tipo XMLType.

Como exemplo, considere as seguintes consultas sobre a relação CLIENTES_REL:

```
SELECT XMLElement("cliente",
          XMLAttributes(C.CCODIGO AS "codigo"),
          XMLElement("nome", C.CNOME),
          XMLForest(C.CFONE1 AS "fone",
                    C.CFONE2 AS "fone",
                    C.CFONE2 AS "fone"),
          (SELECT XMLAgg( XMLElement("pedido",
                     XMLElement("codigo", P.PCODIGO),
                     XMLElement("data", P.PDATA_PEDIDO) ) )
           FROM PEDIDOS_REL P
           WHERE P.PCLIENTE = C.CCODIGO ) )
FROM CLIENTES_REL C;
```

e

```
SELECT XMLElement("cliente",
          XMLAttributes(C.CCODIGO AS "codigo"),
          XMLElement("nome", C.CNOME),
          XMLForest(C.CFONE1 AS "fone",
                    C.CFONE2 AS "fone",
                    C.CFONE2 AS "fone"),
          XMLAgg(
             XMLElement("pedido",
                XMLElement("codigo", P.PCODIGO),
                XMLElement("data", P.PDATA_PEDIDO) ) ) )
FROM CLIENTES_REL C, PEDIDOS_REL P
WHERE P.PCLIENTE = C.CCODIGO
GROUP BY C.CCODIGO, C.CNOME, C.CFONE1, C.CFONE2, C.CFONE3,
         P.PCODIGO, P.PDATA_PEDIDO;
```

O resultado de ambas é o mesmo, dado por:

```
<cliente codigo="193">
   <nome>Bryan Huston</nome>
   <fone>+91 11 012 4813</fone>
   <fone>+91 11 083 4813</fone>
   <fone>+91 11 012 4827</fone>
   <pedido>
      <codigo>405</codigo>
      <data>01/07/05</data>
   </pedido>
</cliente
<cliente codigo="195">
   <nome>Cary Stockwell</nome>
   <fone>+91 11 012 4835</fone>
   <pedido>
      <codigo>407</codigo>
      <data>29/06/05</data>
   </pedido>
   <pedido>
      <codigo>408</codigo>
      <data>01/07/05</data>
   </pedido>
</cliente
```

Na primeira consulta, é retornado um agregado XML para todas as cláusulas da subconsulta. Na segunda consulta, é retornado um agregado XML para cada agrupamento definido pela cláusula GROUP BY.

Mais detalhes do padrão podem ser encontrados em [9][10].

2.4 Visões XML

Uma visão XML virtual é uma coleção XML cujo estado é computado dinamicamente a partir do estado do esquema base sobre o qual ela é definida.

Com a introdução do tipo de dados XML e o padrão SQL/XML, os usuários podem criar uma visão de instâncias de um tipo XML sobre tabelas relacionais usando as funções de publicação do SQL/XML [1], como XMLElement (), XMLConcat (), etc.

Considere, por exemplo, o esquema relacional PEDIDOS_BD e o tipo XML TPedido_XML, cujas representações gráficas são mostradas nas Figuras 2.9 e 2.10, respectivamente. Para gerar instâncias de TPedido_XML a partir de PEDIDOS_BD, definimos a visão SQL/XML mostrada na . Como ilustrado nas Figuras 2.12 e 2.13, para cada tupla da tabela *PEDIDOS_REL*, a visão XML usa as funções de publicação do padrão SQL/XML para construir uma instância do tipo XML TPedido_XML.

Mais detalhadamente, considere o estado da base de dados PEDIDOS_DB mostrado na . O estado correspondente de Pedidos_XML é mostrado na . Este estado da

visão contem uma seqüência de elementos <*Pedido*> do tipo TPedido_XML, que são os elementos primários da visão. Cada elemento <*Pedido*> é construído a partir de uma tupla da tabela de *PEDIDOS_REL* usando a função de publicação XMLElement() do SQL/XML. A função XMLElement() recebe como parâmetro um nome de elemento, uma coleção opcional dos atributos, e zero ou mais argumentos adicionais que compõem o conteúdo do elemento.

Os sub-elementos e atributos de <*Pedido*> são construídos através de subconsultas SQL/XML. Por exemplo,

- O atributo *ID* é construído usando o subconsulta na linha 4. A função XMLAttributes() produz, a partir dos seus argumentos, os atributos do elemento gerado pela função XMLElement() a qual pertence. Estes argumentos são expressões do valor a serem avaliadas, com *alias* opcionais. O tipo de dados de uma expressão de valor de atributo não pode ser um tipo de objeto ou uma coleção. Se uma expressão de valor de atributo resultar em nulo, então nenhum atributo correspondente será criado.
- O sub-elemento <*Date*> é construído usando o subconsulta na linha 5. A função XMLForest() produz uma floresta de elementos XML a partir de seus argumentos, que são expressões a serem avaliadas, com *alias* opcionais. Se uma expressão resultar em nulo, nenhum elemento correspondente será criado.
- O sub-elemento <*Item*> é construído pela subconsulta nas linhas 18-29. A função XMLAgg() é uma função de agregação que produz uma floresta de elementos XML a partir de uma coleção de elementos XML onde os argumentos nulos não aparecem no resultado. Nas linhas 18-29, temos que, para cada tupla na tabela *PEDIDOS_REL*, as tuplas relevantes da tabela *ITENS_REL* são recuperadas e convertidas em uma seqüência de elementos <*Item*>.

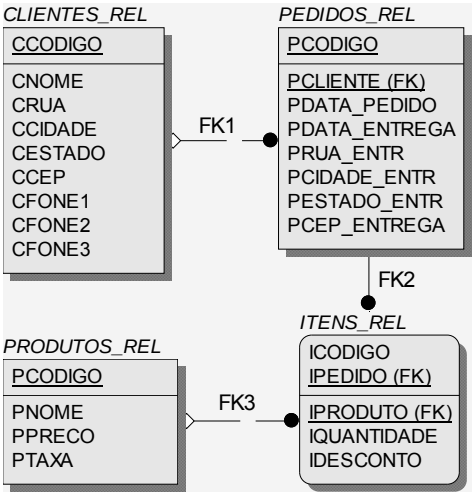

Figura 2.9. Esquema Relacional PEDIDOS_DB

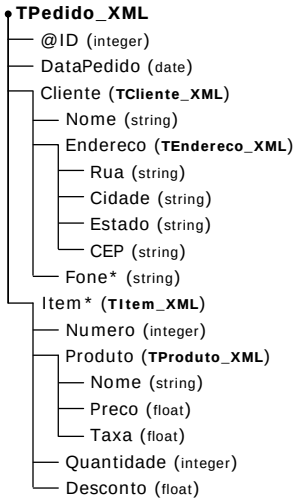

Figura 2.10. Tipo XML TPedido_XML

```
1.  CREATE OR REPLACE VIEW PedidosPorData_XML OF XMLTYPE
2.  XMLSCHEMA "Pedido.xsd" ELEMENT "Pedido"
3.  AS SELECT XMLELEMENT("Pedido",
4.    XMLATTRIBUTES(P.PCODIGO AS "ID"), .....de φ1        τ[PEDIDOS_REL→ TPedido_XML]
5.    XMLFOREST(P.PDATA_ENTREGA AS "DataPedido"), .de φ2
6.    (SELECT XMLELEMENT("Cliente", .....de φ3
7.      XMLFOREST(C.CNOME AS "NOme"), .....de φ4        τ[CLIENTES_REL→ TCliente_XML]
8.      XMLELEMENT("Endereco", .....de φ5
9.        XMLFOREST(C.CRUA AS "Rua"), .....de φ6        τ[CLIENTES_REL→ TEndereco_XML]
10.       XMLFOREST(C.CCIDADE AS "Cidade"), .....de φ7
11.       XMLFOREST(C.CESTADO AS "Estado"), .....de φ8
12.       XMLFOREST(C.CCEP AS "CEP") ), .....de φ9
13.     XMLFOREST(C.CFONE1 AS "Fone"), .....de φ10
14.     XMLFOREST(C.CFONE2 AS "Fone"),
15.     XMLFOREST(C.CFONE3 AS "Fone") )
16.   FROM CLIENTES_REL C
17.   WHERE C.CCODIGO = P.PCLIENTE),
18.   (SELECT XMLAGG( XMLELEMENT("Item", .....de φ11
19.     XMLFOREST(I.ICODIGO AS "Numero"), .....de φ12   τ[ITENS_REL→ TItem_XML]
20.     (SELECT XMLELEMENT("Produto", .....de φ13
21.       XMLFOREST(D.PNOME AS "Nome"), .....de φ14     τ[PRODUTOS_REL→ TProduto_XML]
22.       XMLFOREST(D.PPRECO AS "Preco"), .....de φ15
23.       XMLFOREST(D.PTAXA AS "Taxa") ) .....de φ16
24.     FROM PRODUTOS_REL D
25.     WHERE D.PCODIGO = I.IPRODUTO),
26.     XMLFOREST(I.IQUANTIDADE AS "Quantidade"),..de φ17
27.     XMLFOREST(I.IDESCONTO AS "Desconto") ) ) ....de φ18
28.   FROM ITENS_REL I
29.   WHERE I.IPEDIDO = P.PCODIGO ) )
30. FROM PEDIDOS_REL P
```

Figura 2.11. Definição SQL/XML da visão Pedido_XML

CLIENTES_REL

CCODIGO	CNOME	RUA	CIDADE	ESTADO	CEP	FONE1	FONE2	FONE3
193	Bryan Huston	8 Automation Ln	Albany	NY	12205	+91 11 012 4813	+91 11 083 4813	+91 33 012 4827
195	Cary Stockwell	400 E Joppa Rd	Baltimore	MD	21286	+91 11 012 4835	NULL	NULL

PRODUTOS_REL

PCODIGO	NOME	PRECO	TAXA
2638	HD 10GB 5400	125.50	0.01
1721	PC Bag - L/S	256.28	0.005
1761	Mouse +WP/CL	32.89	0.0

ITENS_REL

IPEDIDO	ICODIGO	IPRODUTO	QUANTIDADE	DESCONTO
405	1	2638	35	0.07
407	1	1721	15	0.05
408	1	1721	30	0.05
407	2	1761	60	0.10

PEDIDOS_REL

PCODIGO	PCLIENTE	PDATA_PEDIDO	PDATA_ENTREGA	PRUA_ENTR	PCIDADE_ENTR	PESTADO_ENTR	PCEP_ENTR
405	193	01/07/05	05/07/05	8 Automation Ln	Albany	NY	12205
407	195	29/06/05	01/07/05	400 E Joppa Rd	Baltimore	MD	21286
408	195	01/07/05	06/07/05	23985 Bedford Rd N	Battle Creek	MI	49017

Figura 2.12. Um estado de PEDIDOS_DB

```
<Pedido ID="405">
  <DataEntrega>2005-07-01</DataEntrega>
  <Cliente>
    <Nome>Bryan Huston</Nome>
    <Endereco>
      <Rua>8 Automation Ln</Rua>
      <Cidade>Albany</Cidade>
      <Estado>NY</Estado>
      <CEP>12205</CEP>
    </Endereco>
    <Fone>+91 11 012 4813</Fone>
    <Fone>+91 11 083 4813</Fone>
    <Fone>+91 33 012 4827</Fone>
  </Cliente>
  <Item>
    <Numero>1</Numero>
    <Produto>
      <Nome>HD 10GB 5400</Nome>
      <Preco>125,5</Preco>
      <Taxa>0.01</Taxa>
    </Produto>
    <Quantidade>35</Quantidade>
    <Desconto>0.01</Desconto>
  </Item>
</Pedido>
```

```
<Pedido ID="408">
  <DataEntrega>2004-05-24</DataEntrega>
  <Cliente>
    <Nome>Cary Stockwell</Nome>
    <Endereco>
      <Rua>400 E Joppa Rd</Rua>
      <Cidade>Baltimore</Cidade>
      <Estado>MD</Estado>
      <CEP>21286</CEP>
    </Endereco>
    <Fone>+91 11 012 4835</Fone>
    <Fone>+91 11 083 4835</Fone>
  </Cliente>
  <Item>
    <Numero>1</Numero>
    <Produto>
      <Nome>PC Bag - L/S</Nome>
      <Preco>256,28</Preco>
      <Taxa>0.005</Taxa>
    </Produto>
    <Quantidade>30</Quantidade>
    <Desconto>0.05</Desconto>
  </Item>
</Pedido>
```

Figura 2.13. Um estado da visão Pedido_XML

2.5 Assertivas de Correspondência de Caminho

Nesta seção, apresentamos o nosso formalismo para especificação de visões XML sobre bases de dados relacionais. Sejam R, R_1,...,R_n esquemas de relação de uma esquema relacional S. Sejam R, R_1,...,R_n relações sobre R, R_1,...,R_n, respectivamente.

Definição 2.1: Seja fk uma chave estrangeira de R_1 que referencia R_2. Dizemos que:

i) fk é um *link* de R_1 para R_2.

ii) fk^{-1}, a *inversa* de fk, é um *link* de R_2 para R_1. □

Definição 2.2:

i) Seja ℓ um link de R_1 para R_2 da forma $R_1[a_1,...,a_m] \subseteq R_2[b_1,...,b_m]$. Seja r_1 uma tupla de R_1. Então, $r_1/\ell = \{ r_2 \in R_2 \mid r_1.a_i = r_2.b_i$, para $1 \leq i \leq m \}$.

ii) Seja ℓ um link de R_2 para R_1 da forma $R_1[a_1,...,a_m] \subseteq R_2[b_1,...,b_m]$. Seja r_2 uma tupla de R_2. Então, $r_2/\ell = \{ r_1 \in R_1 \mid r_1.a_i = r_2.b_i$, para $1 \leq i \leq m \}$. □

Definição 2.3: Seja ℓ um link de R_1 para R_2, e seja r_1 e r_2 tuplas de R_1 e R_2, respectivamente. Dizemos que:

i) r_1 *referencia* r_2 *através de* ℓ sse $r_2 \in r_1/\ell$.

ii) ℓ *tem ocorrência simples* sse uma tupla de R_1 pode referenciar pelo menos uma tupla de R_2 através de ℓ; caso contrário, ℓ *tem ocorrência múltipla*. □

Definição 2.4: Sejam ℓ_1,... ,ℓ_n links. Assuma que:

i) ℓ_1 é uma chave estrangeira de R da forma $R[a_1{}^{\ell}1,...,a_{m1}{}^{\ell}1] \subseteq R_1[b_1{}^{\ell}1,...,b_{m1}{}^{\ell}1]$ ou a inversa de uma chave estrangeira de R_1 da forma $R_1[b_1{}^{\ell}1,..., b_{m1}{}^{\ell}1] \subseteq R[a_1{}^{\ell}1,...,a_{m1}{}^{\ell}1]$

ii) ℓ_i é uma chave estrangeira de R_{i-1} da forma $R_{i-1}[a_1{}^{\ell}i,..., a_{mi}{}^{\ell}i] \subseteq R_i[b_1{}^{\ell}i,...,b_{mi}{}^{\ell}i]$ ou a inversa de uma chave estrangeira de R_i da forma $R_i[b_1{}^{\ell}i,.., b_{mi}{}^{\ell}i] \subseteq R_{i-1}[a_1{}^{\ell}i, ..., a_{mi}{}^{\ell}i]$, para $2 \leq i \leq n$.

Dizemos que:

i) $\varphi = \ell_1. ... .\ell_n$ é uma *caminho referencial* de R para R_n.

ii) as tuplas de R *referenciam tuplas de* R_n *através de* φ.

iii) φ *tem ocorrência simples* sse ℓ_i *tem ocorrência simples*, para $1 \leq i \leq n-1$; caso contrário, φ *tem ocorrência múltipla*. □

Definição 2.5: Seja $\varphi = \ell_1. \ldots .\ell_n$ um caminho referencial de R para R_n. Seja r uma tupla de R. Então,

$$r / \varphi = \{ r_n \in R_n \mid (\exists r_1 \in R_1) \ldots (\exists r_{n-1} \in R_{n-1}) \; (r.a_k^{\ell 1} = r_1.b_k^{\ell 1}, \text{ para } 1 \leq k \leq m_1)$$

$$\text{e } (r_{i-1}.a_k^{\ell i} = r_i.b_k^{\ell i}, \text{ para } 1 \leq k \leq m_i \text{ e } 2 \leq i \leq n) \}. \square$$

Definição 2.6: Um *caminho* de R é uma expressão de uma das seguintes formas:

i) NULL

ii) a, onde a é um atributo de R.

iii) $\{a_1,\ldots,a_n\}$, onde $a_1,\ldots,a_n$ são atributos de R.

iv) $\varphi.a$, onde φ é um caminho referencial de R para R' e a é um atributo de R'.

v) $\varphi.\{a_1,\ldots,a_n\}$, onde φ é um caminho referencial de R para R' e $a_1,\ldots,a_n$ são atributos de R'. $\square$

Definição 2.7: Seja r uma tupla de R.

i) $r / \text{NULL} = \{ r \}$.

ii) $r / a = \{ v \mid v = r.a \text{ e } v \neq NULL \}$, onde a é um atributo de R.

iii) $r / \{a_1,\ldots,a_m\} = \{ v \mid v = r.a_i \text{ com } 1 \leq i \leq m \text{ e } v \neq NULL \}$, onde $a_1,\ldots,a_n$ são atributos de R.

iv) $r / \varphi.a = \{ v \mid \exists r' \in r / \varphi \text{ e } v \in r'/a \}$, onde φ é um caminho referencial de R para R', a é um atributo de R' e r' é uma tupla de R'.

v) $r / \varphi.\{a_1,\ldots,a_m\} = \{ v \mid \exists r' \in r / \varphi \text{ e } v \in r'/\{a_1,\ldots,a_m\} \}$, onde φ é um caminho referencial de R para R', $a_1,\ldots,a_n$ são atributos de R' e r' é uma tupla de R'. $\square$

Definição 2.8: Dizemos que um tipo complexo T do XML Schema é *restrito* sse T é definido usando somente os construtores *complexType* e *sequence*, e o tipo de seus atributos é um tipo XML simples.

No restante desta seção, seja T um tipo complexo restrito, e sejam R e R' esquemas de relação de um esquema relacional S.

Definição 2.9: Uma *assertiva de correspondência (AC)* é uma expressão da forma $[T/e] \equiv [R/\delta]$ onde e é um elemento ou um atributo de T, de tipo T_e, e δ é um caminho de R tal que:

i) Se e é um atributo ou um elemento de ocorrência simples e T_e é um tipo simples, então δ tem uma das seguintes formas:

- a, onde a é um atributo de R cujo tipo é compatível com T_e;
- $\varphi.a$, onde φ é um caminho referencial de R para R' tal que φ tem ocorrência simples, e a é um atributo de R' cujo tipo é compatível com T_e.

ii) Se e é um elemento de ocorrência múltipla e T_e é um tipo simples, então δ tem uma das seguintes formas:
   - φ.a, onde φ é um caminho referencial de *R* para *R'* tal que φ tem ocorrência múltipla e a é um atributo de *R'*, cujo tipo é compatível com T_e;
   - $\{a_1,...,a_n\}$, onde $a_1,...,a_n$ são atributos de *R* tais que o tipo de a_i é compatível com T_e, para $1 \leq i \leq n$;
   - φ.$\{a_1,...,a_n\}$, onde φ é um caminho referencial de *R* para *R'* tal que φ tem ocorrência simples, e $a_1,...,a_n$ são atributos de *R'* tais que o tipo de a_i é compatível com T_e, para $1 \leq i \leq n$.

iii) Se e é um elemento de ocorrência simples e T_e é um tipo complexo, então δ tem uma das seguintes formas:
   - φ, onde φ é um caminho referencial de *R* para *R'* tal que φ tem ocorrência simples;
   - NULL

iv) Se e é um elemento de ocorrência múltipla e T_e é um tipo complexo, então δ é um caminho de *R* para *R'* tal que δ tem ocorrência múltipla. □

Definição 2.10: Seja $\mathcal{A}$ um conjunto de assertivas de correspondência. Dizemos que $\mathcal{A}$ *especifica completamente* T *em termos de R* sse

i) Para cada elemento ou atributo e de T, existe uma única AC em $\mathcal{A}$ da forma [T/e] ≡ [*R*/δ], chamada *a AC para* e *em* $\mathcal{A}$.

ii) Para cada assertiva em $\mathcal{A}$ da forma [T/e] ≡ [*R*/δ], onde e é um elemento do tipo complexo T_e e δ é um caminho referencial de *R* para *R'*, então $\mathcal{A}$ especifica completamente T_e em termos de *R'*.

iii) Para cada assertiva em $\mathcal{A}$ da forma [T/e] ≡ [*R*/NULL], onde e é um elemento do tipo complexo T_e, então $\mathcal{A}$ especifica completamente T_e em termos de *R*. □

Definição 2.11: Seja $\mathcal{A}$ um conjunto de assertivas de correspondência tal que $\mathcal{A}$ especifica completamente T em termos de *R*. Seja R uma relação sobre *R*.

i) Suponha que T seja um tipo XML simples. Seja S_1 um conjunto de elementos de tipo T. Seja S_2 um conjunto de valores de um tipo escalar do SQL. Dizemos que $S_1 \equiv_{\mathcal{A}} S_2$ sse

$$\$t \in S_1 \text{ sse existe } \boldsymbol{v} \in S_2 \text{ tal que } \$t / \text{text}() = f(\boldsymbol{v})$$

onde f é uma função que mapeia um valor SQL em um valor XML [10].

ii) Suponha que T seja um tipo XML simples. Seja S_1 um conjunto de valores de tipo T. Seja S_2 um conjunto de valores de um tipo escalar do SQL. Dizemos que $S_1 \equiv_{\mathcal{A}} S_2$ sse

$$v_1 \in S_1 \text{ sse existe } v_2 \in S_2 \text{ tal que } v_1 = f(v_2)$$

onde f é uma função que mapeia um valor SQL em um valor XML [10].

iii) Suponha que T seja um tipo XML complexo. Seja S_1 um conjunto de elementos de tipo T. Seja S_2 um conjunto de tuplas de R. Dizemos que $S_1 \equiv_{\mathcal{A}} S_2$ sse

$$\$t \in S_1 \text{ sse existe } r \in S_2 \text{ tal que } \$t \equiv_{\mathcal{A}} r.$$

iv) Seja r uma tupla de R e seja $t uma instância de T. Dizemos que $t $\equiv_{\mathcal{A}}$ r sse, para cada elemento e de T tal que [T/e] ≡ [R/δ] é a AC para e em $\mathcal{A}$ (a qual existe em $\mathcal{A}$ por suposição), então $t/e $\equiv_{\mathcal{A}}$ r/δ, e, para cada atributo a de T tal que [T/a] ≡ [R/δ] é a AC para a em $\mathcal{A}$ (a qual existe em $\mathcal{A}$ por suposição), então DATA($t/@a) $\equiv_{\mathcal{A}}$ r/δ.

Se $t $\equiv_{\mathcal{A}}$ r, dizemos que $t é *semanticamente equivalente a* r *como especificado por* $\mathcal{A}$. □

2.6 Usando Assertivas de Correspondência para Especificar Visões XML

Propomos especificar uma visão XML com auxílio de um conjunto de assertivas de correspondência, as quais especificam de forma axiomática como os elementos da visão são sintetizados a partir de tuplas da base relacional. Seja S um esquema relacional. Uma *Visão XML* sobre S é uma quádrupla V = <e, T, ψ, $\mathcal{A}$>, onde:

(i) e é o nome do elemento primário da visão;

(ii) T é o tipo XML de e;

(iii) ψ é uma assertiva de correspondência global da forma [V]≡[*R*], onde *R* é um esquema de relação ou um esquema de visão relacional de S.

(iv) $\mathcal{A}$ é um conjunto de assertivas de correspondência de caminho que especificam completamente T em termos de *R*.

Dizemos que o par <e, T> é o *esquema de* V e *R* é o *esquema da relação (ou visão) pivô* da visão XML, tal que existe mapeamento 1-1 entre as tuplas da tabela/visão pivô e os elementos da visão. Como discutido em [12], caso não haja relação ou visão que satisfaça a restrição do mapeamento 1-1, devemos primeiro definir uma visão relacional que satisfaça essa restrição;

Considere, por exemplo, a visão Pedidos_XML, cujo elemento primário <*Pedido*> tem tipo TPedido_XML, e cujo esquema de relação pivô é *PEDIDOS_REL*. mostra a especificação SQL/XML de Pedidos_XML e mostra a representação gráfica de TPedido_XML. Figura 2.14 mostra as assertivas de correspondência de Pedidos_XML, as quais especificam completamente TPedido_XML em termos de *PEDIDOS_REL*.

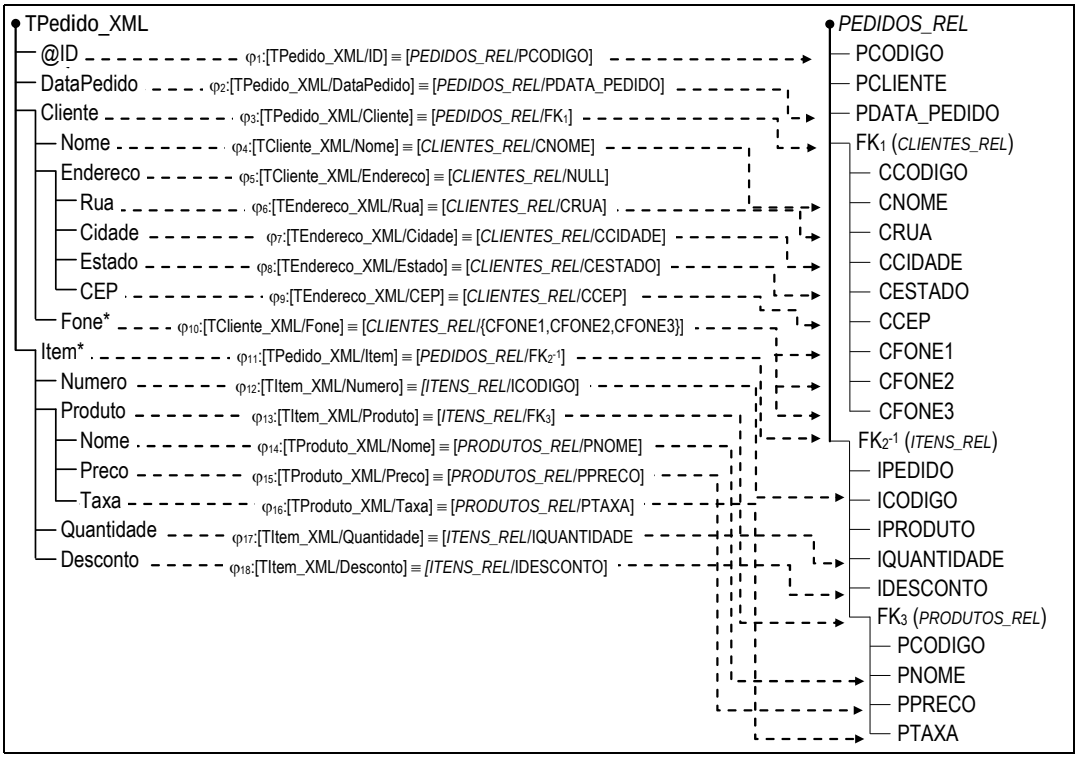

Figura 2.14. Assertivas de Correspondência da visão Pedidos_XML

Desenvolvemos uma ferramenta (vide Seção 2.8), chamada *XML View-By-Assertions* (XVBA), para suportar a definição das assertivas de correspondência da visão (Assertiva Global e Assertivas de Correpondência de Caminho da visão). XVBA apresenta uma interface gráfica simples que permite aos usuários navegar para tabelas relacionadas à tabela pivô. O processo começa com o usuário carregando o esquema da visão e o esquema do banco na ferramenta. Em seguida, o usuário pode conectar graficamente elementos/atributos do tipo da visão com atributos/caminhos da relação pivô.

O processo de geração das assertivas de correspondência de Pedidos_XML através da ferramenta XVBA é realizado da seguinte forma: (1) associando os elementos/atributos de TPedido_XML com atributos/caminhos de *PEDIDOS_REL*; e (2) descendo recursivamente para os sub-elementos de TPedido_XML para definir as assertivas destes. Por exemplo, para definir a assertiva do elemento Item, (φ_{11}:[TPedido_XML/Item] ≡ [*PEDIDOS_REL*/FK2^{-1}]), o usuário seleciona o elemento Item no esquema da visão e a chave estrangeira inversa FK2^{-1} no esquema do banco.

2.7 Geração Automática da Definição SQL/XML a partir das Assertivas da Visão

Seja S um esquema relacional e V = <e, T, R, $\mathcal{A}$> uma visão XML sobre S. Nesta seção, mostramos que as assertivas de correspondência em $\mathcal{A}$ definem um mapeamento que pode ser corretamente traduzido para uma consulta em SQL/XML que define a visão. É importante notar que o formalismo das assertivas de correspondência é independente de tecnologia, de modo que a implementação de outras definições de visão pode ser facilmente adaptada.

Dado um estado σ_S de S, denotamos por $\sigma_S(R)$ a relação que σ_S associa a R. Além disso, dado um elemento e, o *conteúdo estendido* de e é uma lista de atributos e elementos filhos de e. As assertivas de correspondência em $\mathcal{A}$ definem um mapeamento funcional, denotado DEF_V, de instâncias do esquema base S para instâncias do esquema da visão. Dado um estado σ_S de S, o valor de V em σ_S de S é dado por:

$DEF_V(\sigma_S)$ = { $t | $t é um elemento <$e$> de tipo T e $\exists r \in \sigma_S(R)$ tal que $t $\equiv_{\mathcal{A}}$ r }.

A definição SQL/XML de V é dada por:

```
CREATE VIEW V
AS SELECT XMLElement( "e", τ[R→T][r] )
    FROM R r
```

onde $\tau[R \rightarrow T][r]$ é uma seqüência de subconsultas SQL/XML, uma para cada elemento/atributo de T. Dada uma tupla r de $\sigma_S(R)$, $\sigma_S(\tau[R \rightarrow T][r])(r)$ denota o resultado de se avaliar as subconsultas SQL/XML no estado σ_S, com r substituído por r. Provaremos que, dada uma instância $t de T cujo conteúdo estendido é construído a partir de $\sigma_S(\tau[R \rightarrow T][r])(r)$, então $t $\equiv_{\mathcal{A}}$ r.

Figura 2.15 mostra o algoritmo GeraConstrutorSQL/XML que gera a função construtora $\tau[R \rightarrow T][r]$. Figura 2.16 apresenta o algoritmo GeraSubconsultaSQL/XML, onde δ é um caminho da forma $\varphi_1. \ldots . \varphi_n$, como definido em Definição 2.6, e Join$\varphi(r)$ é definido pelo seguinte fragmento SQL:

$R_1\ r_1, \ldots, R_n\ r_n$

WHERE $r.a_1^{\ell_1} = r_1.b_1^{\ell_1}$ AND … AND $r.a_{m_1}^{\ell_1} = r_1.b_{m_1}^{\ell_1}$

AND $r_1.a_1^{\ell_2} = r_2.b_1^{\ell_2}$ AND … AND $r_1.a_{m_2}^{\ell_2} = r_2.b_{m_2}^{\ell_2}$

…

AND $r_{n-1}.a_1^{\ell_n} = r_n.b_1^{\ell_n}$ AND … AND $r_1.a_{m_n}^{\ell_n} = r_2.b_{m_n}^{\ell_n}$

Input: um tipo XML T, um esquema de relação *R*, um conjunto de assertivas de correspondência $\mathcal{A}$ que especifica completamente T em termos de *R* e *r* é um alias para *R*.
Output: Função $\tau[R \rightarrow T][r]$.

```
Seja τ uma string;
τ := ∅;
Se T tem atributos Então
  τ := τ + "XMLAttributes("
  Para cada atributo a de tipo T onde φa é a AC para a em A Faça
      τ := τ + GeraSubconsultaSQL/XML(A, φa, r);
  Fim Para;
  τ := τ + ")"
Fim Se;
Para Cada elemento e de T onde φe é a AC para e em A Faça
      τ := τ + GeraSubconsultaSQL/XML(A, φe, r);
Fim Para;
Retorne τ ;
```

Figura 2.15. Algoritmo GeraConstrutorSQL/XML

Input: um conjunto $\mathcal{A}$ de assertivas de correspondência que especificam completamente T em termos de *R*, uma AC $[T/e] \equiv [R/\delta]$ em $\mathcal{A}$ onde e é um elemento ou atributo de tipo T_e, e um alias *r* para *R*
Output: uma sub-consulta SQL/XML

```
Seja Q uma string;
No caso
    Caso 1: Se e é um elemento de ocorrência simples, Te é um tipo simples e δ = a, Então
            Q := "XMLFOREST(r.a AS \"e\")";
    Caso 2: Se e é um elemento de ocorrência simples, Te é um tipo simples e δ = φ.a, Então
            Q := "XMLFOREST( (SELECT rn.a FROM Joinφ(r) ) AS \"e\")";
    Caso 3: Se e é um elemento de ocorrência múltipla, Te é um tipo simples e δ = {a1,...,an}, Então
            Q := "XMLCONCAT( XMLFOREST(r.a1 AS \"e\")," + ...+ "XMLFOREST(\"e\", r.an AS \"e\") )";
    Caso 4: Se e é um elemento de ocorrência múltipla, Te é um tipo simples δ = φ/ {a1,...,an}, Então
            Q := "XMLCONCAT( (SELECT XMLFOREST( rn.a1 AS \"e\",  ... ,  rn.an AS \"e\" ) FROM Joinφ(r) ) )";
    Caso 5: Se e é um elemento de ocorrência múltipla, Te é um tipo simples δ = φ/ a, Então
            Q := "(SELECT XMLAGG( XMLFOREST( rn.a AS \"e\" ) FROM Joinφ(r) )";
    Caso 6: Se e é um elemento de ocorrência simples, Te é um tipo complexo e δ = φ, Então
           Q := "(SELECT XMLELEMENT(\"e\", " + GeraConstrutorSQL/XML(Te, Rn, A, rn) + ") FROM Joinφ(r) ) ";
    Caso 7: Se e é um elemento de ocorrência múltipla, Te é um tipo complexo δ = φ, Então
            Q := "(SELECT XMLAGG( XMLELEMENT(\"e\",
                + GeraConstrutorSQL/XML(Te, Rn, A, rn) + ") ) FROM Joinφ(r) )";
    Caso 8: Se e é um elemento de ocorrência simples, Te é um tipo complexo δ = NULL, Então
            Q := "XMLELEMENT(\"e\", " + GeraConstrutorSQL/XML(Te, R, A, r) + ")";
    Caso 9: Se e é um atributo e δ = a, Então
            Q := " r.a AS \"e\" ";
    Caso 10: Se e é um atributo e δ = φ.a, Então
            Q := " (SELECT rn.a FROM Joinφ(r) ) AS \"e\" ";
Fim Caso;
Retorne Q ;
```

Figura 2.16. Algoritmo GeraSubconsultaSQL/XML

2.7.1 Corretude dos Algoritmos GeraConstrutorSQL/XML e GeraSubconsultaSQL/XML

A corretude desses algoritmos segue das proposições e teoremas abaixo. No que se segue, seja T um tipo XML, R um esquema de relação, $\mathcal{A}$ um conjunto de assertivas de correspondência que especifica completamente T em termos de R, e r um alias para R.

Proposição 2.1: Seja φ a AC $[T/e] \equiv [R/\delta]$ em $\mathcal{A}$ para o elemento e de tipo T_e. Seja GeraSubconsultaSQL/XML($\mathcal{A}$, φ, r) = $Q_e[r]$. Seja r uma tupla de $\sigma_S(R)$ e seja $\mathcal{S}$ o conjunto de elementos <e> resultante da execução de $Q_e[r]$ sobre σ_S com r trocado por r. Então, temos que $\mathcal{S} \equiv_{\mathcal{A}} r/\delta$. □ (Veja a prova em [47]).

Proposição 2.2: Seja φ a AC $[T/a] \equiv [R/\delta]$ em $\mathcal{A}$ para o atributo a de tipo T_a. Seja GeraSubconsultaSQL/XML($\mathcal{A}$, φ, r) = $Q_a[r]$. Seja r uma tupla de $\sigma_S(R)$ e seja v_a o valor resultante da execução de $Q_a[r]$ sobre σ_S com r trocado por r. Então, temos que $v_a = f(v)$, onde v é o único valor em r/δ, e f é a função que mapeia valores SQL em valores XML [10]. □ (Veja a prova em [47]).

Teorema 2.1:

Sejam $a_1,...,a_k$ atributos de T e sejam $e_1,...,e_m$ elementos de T.

Seja GeraConstrutorSQL/XML(R, r, T, $\mathcal{A}$) = $\tau[R \rightarrow T][r]$.

Seja r uma tupla de $\sigma_S(R)$.

Seja $t um elemento <e> de tipo T cujo conteúdo estendido é construído a partir de $\sigma_S(\tau[R \rightarrow T][r])(r)$.

Então, $t $\equiv_{\mathcal{A}}$ r. □

Prova: Sejam $a_1,...,a_k$ atributos de T. Seja φ_{a_i} a AC para a_i em $\mathcal{A}$ e seja T_{a_i} o tipo de a_i, para $1 \leq i \leq k$. Suponha que φ_{a_i} é da forma $[T/a_i] \equiv [R/\delta_{a_i}]$. Sejam $e_1,...,e_m$ elementos de T. Seja φ_{e_i} a AC para e_i em $\mathcal{A}$ e seja T_{e_i} o tipo de e_i, para $1 \leq i \leq m$. Suponha que φ_{e_i} é da forma $[T/e_i] \equiv [R/\delta_{e_i}]$. Seja $\tau[R \rightarrow T][r]$ a função construtora gerada pelo algoritmo GeraConstrutorSQL/XML. Do algoritmo, temos que:

$\tau[R \rightarrow T][r]$ = XMLAttributes($Qa_1[r]$, ... $Qa_k[r]$), $Qe_1[r]$, ... $Qe_m[r]$, onde

$Q_{a_i}[r]$ = GeraSubconsultaSQL/XML($\mathcal{A}$, φ_{a_i}, r), para $1 \leq i \leq k$

$Q_{e_i}[r]$ = GeraSubconsultaSQL/XML($\mathcal{A}$, φ_{e_i}, r), para $1 \leq i \leq m$

Seja r uma tupla de $\sigma_S(R)$. Seja $t um elemento <e> de T cujo conteúdo estendido é construído a partir de $\sigma_S(\tau[R \rightarrow T][r])(r)$. Para $1 \leq i \leq k$, seja v_{a_i} o valor resultante da execução

de $Q_{ai}[r]$ sobre σ_S com *r* substituído por r. Para $1 \leq i \leq m$, seja $\mathcal{S}_{ei}$ um conjunto de elementos $<e_i>$ resultante da execução de $Q_{ei}[r]$ em σ_S com *r* trocado por r. Consequentemente, $t = <e a_1 ="v_{a1}" ... a_k ="v_{ak}"> $\mathcal{S}_{e1}$ ... $\mathcal{S}_{em}$ </e>. Da Proposição 2.1, temos que $\mathcal{S}_{ei} \equiv_{\mathcal{A}} r/\delta_{ei}$, para $1 \leq i \leq m$. Da Proposição 2.2, temos que $v_{ai} = f(v)$, para $1 \leq i \leq k$, onde v é o único valor em r/δ_{ai}, e f é uma função que mapeia valores SQL em valores XML. Então, da Definição 2.11(ii), temos que DATA($t/@$a_i$) $\equiv_{\mathcal{A}} r/\delta_{ai}$, $1 \leq i \leq k$. Portanto, da Definição 2.11(iv), temos que $t $\equiv_{\mathcal{A}}$ r. □

2.7.2 Corretude da Função Construtora τ[*PEDIDOS_REL*→TPedido_XML]

No que segue, para simplificar, considere que $\tau[R \rightarrow T](r)$ denota a função $\sigma_S(\tau[R \rightarrow T][r])(r)$ que constrói o conteúdo estendido de uma instância $t de T tal que $t $\equiv_{\mathcal{A}}$ r.

Considere, por exemplo, a definição SQL/XML de Pedidos_XML, mostrada na . A função construtora τ[*PEDIDOS_REL*→TPedido_XML](O) (linhas 3 a 29) constrói o conteúdo estendido de uma instância de TPedidos_XML a partir de uma tupla de PEDIDOS_REL. A função construtora contém quatro subconsultas, uma para cada elemento e atributo de TPedido_XML. Em GeraSubconsultaSQL/XML, cada subconsulta é gerada a partir da AC do elemento ou atributo correspondente. mostra a assertiva que gerou cada subconsulta SQL/XML de τ[*PEDIDOS_REL*→TPedido_XML](O).

Mostraremos que τ[*PEDIDOS_REL*→TPedido_XML](O) constrói o conteúdo estendido de uma instância $P de TPedido_XML, para cada tupla O de um estado de *PEDIDOS_REL,* tal que $P é semanticamente equivalente a O, como especificado pelas assertivas de Pedidos_XML.

Seja PEDIDOS_DB um estado do esquema relacional PEDIDOS_DB. Sejam CLIENTES_REL, PEDIDOS_REL, PRODUTOS_REL e ITENS_REL estados que PEDIDOS_DB associa com os esquemas de relação *CLIENTES_REL*, *PEDIDOS_REL*, *PRODUTOS_REL* e *ITENS_REL* de PEDIDOS_DB, respectivamente.

Seja O uma tupla de PEDIDOS_REL e seja $P uma instância de TPedido_XML cujo conteúdo estendido é construído com τ[*PEDIDOS_REL*→TPedido_XML](O). Da Definição 2.11 e de φ_1, φ_2, φ_3 e φ_{11}, temos que $P $\equiv_{\mathcal{A}}$ O sse:

(1) DATA($P / @ID) $\equiv_{\mathcal{A}}$ O / PCODIGO,

(2) $P / DataPedido $\equiv_{\mathcal{A}}$ O / PDATA_PEDIDO,

(3) $P / Cliente $\equiv_{\mathcal{A}}$ O / FK_1, e

(4) $P / Item $\equiv_{\mathcal{A}}$ O / FK_2^{-1}.

Prova de (1): Da linha 4 da , temos que:

DATA($P / @ID) = { v | v = f(O.PCODIGO) e $v \neq$ NULL }.

Da Definição 2.7(ii), temos que

O / PCODIGO = { D | D = O.PCODIGO e O.PCODIGO ≠ NULL }.

Então, da Definição 2.11(ii), temos que: DATA($P / @ID) $\equiv_{\mathcal{A}}$ O / PCODIGO. □

Prova de (2): Da linha 5 da , temos que:

$P / DataPedido = { $D | $D=<DataPedido>$f$(O.PDATA_PEDIDO) </DataPedido> e

O.PDATA_PEDIDO ≠ NULL }.

Da Definição 2.7(ii), temos que:

O / PDATA_PEDIDO = { D | D = O.PDATA_PEDIDO e O.PDATA_PEDIDO ≠ NULL }.

Então, da Definição 2.11(i), temos que: $P / DataPedido $\equiv_{\mathcal{A}}$ O / PDATA_PEDIDO. □

Prova de (3): Das linhas 6-17 da , temos que:

$P / Cliente = { $C | ∃ C ∈ CLIENTES_REL tal que

C.CCODIGO = O.PCLIENTE e o conteúdo estendido de $C é

construído de τ[*CLIENTES_REL*→TCliente_XML](C)}.

A seguir, mostramos que, dada uma tupla C ∈ CLIENTES_REL e um elemento $C cujo conteúdo estendido é construído de τ[*CLIENTES_REL*→TCliente_XML](C), então $C $\equiv_{\mathcal{A}}$ C. De φ_4, φ_5, e φ_{10}, temos que $C $\equiv_{\mathcal{A}}$ C sse:

(3.1) $C / Nome $\equiv_{\mathcal{A}}$ C / CNOME,

(3.2) $C / Endereco $\equiv_{\mathcal{A}}$ C / NULL, e

(3.3) $C / Fone $\equiv_{\mathcal{A}}$ C / {CFONE1, CFONE2, CFONE3}.

Prova de (3.1): A prova segue da linha 7 da e é similar à Prova de (2).

Prova de (3.2): Das linhas 8-12 da , temos que:

$C / Endereco = { $A } onde o conteúdo estendido de $A é construído de

τ[*CLIENTES_REL*→TEndereco_XML](C).

A seguir, mostramos que, se o conteúdo estendido de $A é construído de τ[*CLIENTES_REL*→TEndereco_XML](C) então $A $\equiv_{\mathcal{A}}$ C. De φ_6, φ_7, φ_8 e φ_9, temos que $A $\equiv_{\mathcal{A}}$ C sse

(3.2.1) $A / Rua $\equiv_{\mathcal{A}}$ C / CRUA,

(3.2.2) $A / Cidade $\equiv_{\mathcal{A}}$ C / CCIDADE,

(3.2.3) $A / Estado $\equiv_{\mathcal{A}}$ C / CESTADO, e

(3.2.4) $A / CEP $\equiv_{\mathcal{A}}$ C / CCEP.

As provas de (3.2.1), (3.2.2), (3.2.3) e (3.2.4) são similares à Prova de (2) e seguem das linhas 9, 10, 11 e 12 da , respectivamente. Então, de (3.2.1), (3.2.2), (3.2.3) e (3.2.4), temos que $C / Endereco = {$A} onde $A $\equiv_{\mathcal{A}}$ C.

Da Definição 2.7(i), temos que C / NULL = {C}. Consequentemente, da Definição 2.11(iii), temos que $C / Endereco $\equiv_{\mathcal{A}}$ C / NULL. □

Prova de (3.3): Das linhas 13-15 da , temos que

$C / Fone = S1 $\cup$ S2 $\cup$ S3 onde:

S1 = { $H | $H = <Fone> f(C.CFONE1) </Fone> e H $\neq$NULL },

S2 = { $H | $H = <Fone> f(C.CFONE2) </Fone> e H $\neq$NULL }, e

S3 = { $H | $H = <Fone> f(C.CFONE3) </Fone> e H $\neq$NULL }.

Da Definição 2.7 (iii) temos que:

C / {CFONE1, CFONE2, CFONE3} = { H | (H = C.CFONE1 ou H = C.CFONE2
ou H = C.CFONE3) e H $\neq$ NULL }.

Consequentemente, da Definição 2.11 (ii), temos que:

$C / Fone $\equiv_{\mathcal{A}}$ C / {CFONE1, CFONE2, CFONE3}.

Então, de (3.1), (3.2) e (3.3), temos que:

$P / Cliente = { $C | $\exists$ C $\in$ CLIENTES_REL tal que
C.CCODIGO = O.PCLIENTE e $C $\equiv_{\mathcal{A}}$ C }.

Da Definição 2.2(i), temos que:

O / FK_1 = { C | $\exists$ C $\in$ CLIENTES_REL tal que C.CCODIGO = O.PCLIENTE }.

Consequentemente, da Definição 2.11(iii), temos que: $P / Cliente $\equiv_{\mathcal{A}}$ O / FK_1. □

Prova de (4): Das linhas 18-29 da , temos que:

$P / Item = { $L | $\exists$ L $\in$ ITENS_REL tal que L.IPEDIDO = O.PCODIGO e
o conteúdo estendido de $L é construído de
τ[*ITENS_REL*$\rightarrow$TItem_XML](L) }.

A seguir, mostramos que, dada uma tupla L $\in$ ITENS_REL e um elemento $L cujo conteúdo estendido é construído de τ[*ITENS_REL*$\rightarrow$TItem_XML](L), então $L $\equiv_{\mathcal{A}}$ L. De φ_{12}, φ_{13}, φ_{17} e φ_{18}, temos que $L $\equiv_{\mathcal{A}}$ L sse:

(4.1) $L / Numero $\equiv_{\mathcal{A}}$ L / ICODIGO,

(4.2) $L / Produto $\equiv_{\mathcal{A}}$ L / FK_3,

(4.3) $L / Quantidade $\equiv_{\mathcal{A}}$ L / QUANTIDADE, e

(4.4) $L / Desconto $\equiv_{\mathcal{A}}$ L / DESCONTO.

As provas de (4.1), (4.3) e (4.4) são similares à Prova de (2) e seguem das linhas 19, 26 e 27 da , respectivamente.

Prova de (4.2): Das linhas 19-25, temos que

$L / Produto = { $D | $\exists$D $\in$ PRODUTOS_REL tal que D.PCODIGO = L.IPRODUTO e
o conteúdo estendido de $D é construído de
τ[*PRODUTOS_REL*$\rightarrow$ TProduto_XML](D) }.

A seguir, mostramos que, dada uma tupla D $\in$ PRODUTOS_REL e um elemento $D cujo conteúdo estendido é construído de τ[*PRODUTOS_REL*$\rightarrow$ TProduto_XML](D), então $D $\equiv_{\mathcal{A}}$ D. De φ_{14}, φ_{15} e φ_{16}, temos que $D $\equiv_{\mathcal{A}}$ D sse:

(4.2.1) $D / Nome $\equiv_{\mathcal{A}}$ D / PNAME,

(4.2.2) $D / Preco $\equiv_{\mathcal{A}}$ D / PPRECO, e

(4.2.3) $D / Taxa $\equiv_{\mathcal{A}}$ D / PTAXA.

As provas de (4.2.1), (4.2.2) e (4.2.3) são similares à Prova de (2) e seguem das linhas 21, 22 e 23 da , respectivamente. Logo, de (4.2.1), (4.2.2) e (4.2.3), temos que:

$L / Produto = { $D | $\exists$D$\in$PRODUTOS_REL tal que D.PCODIGO = L.IPRODUTO e $D $\equiv_{\mathcal{A}}$ D }.

Da Definição 2.2(i), temos que:

L / FK_3 = { D | $\exists$ D $\in$ PRODUTOS_REL tal que D.PCODIGO = L.IPRODUTO }.

Consequentemente, da Definição 2.11(iii), temos que: $L / Produto $\equiv_{\mathcal{A}}$ L / FK_3.

Logo, de (4.1), (4.2), (4.3) e (4.4), temos que:

$P / Item = { $L | $\exists$ L $\in$ ITENS_REL tal que L.IPEDIDO = O.PCODIGO e $L $\equiv_{\mathcal{A}}$ L }.

Da Definição 2.2(ii), temos que:

O / FK_2^{-1} = { L | $\exists$ L $\in$ ITEMS_REL tal que L.IPEDIDO = O.PCODIGO }.

Consequentemente, da Definição 2.11(iii), temos que: $P / Item $\equiv_{\mathcal{A}}$ O / FK_2^{-1}. □

2.8 Ferramenta XVBA: XML View-by-Assertions

Nesta seção, apresentamos XVBA (*XML View By Assertions*), uma ferramenta para facilitar a tarefa de criação de visões XML sob dados relacionais. Uma visão simplificada da arquitetura de XVBA é mostrada na . Através do componente *GUI* (*Graphical User Interface*), o usuário define os mapeamentos entre o esquema da visão XML e o esquema relacional com a ajuda das assertivas de correspondência.

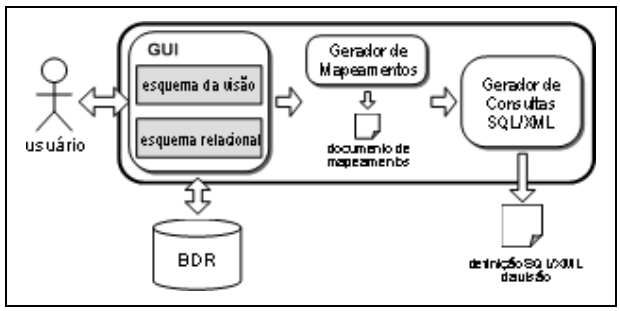

Figura 2.17. Arquitetura de XVBA

O segundo componente, o *Gerador de Mapeamentos*, gera o documento de mapeamentos, o qual contém um conjunto de assertivas de correspondência que especifica completamente o esquema da visão em termos do esquema relacional.

O terceiro componente, o *Gerador de Consulta SQL/XML*, recebe como entrada o documento de mapeamento e gera a definição SQL/XML da visão. Este componente implementa o algoritmo discutido na Seção 2.7.

Mapeamentos entre esquemas têm sido usados para troca [2] e integração de dados [3][5][7]. O sistema *Clio* [3] provê uma forma declarativa de especificar os mapeamentos entre esquemas e usa esses mapeamentos para gerar a consulta que captura semântica dos mapeamentos. Clio permite aos seus usuários especificar um conjunto de mapeamentos entre múltiplos esquemas base e um esquema alvo. Entretanto, o processo de geração de mapeamentos e a geração da consulta são muito mais complexos que o processo com XVBA. Desta forma, XVBA é mais apropriado para a tarefa específica de geração de visão XML sobre uma única base relacional.

O processo para gerar uma visão XML com XVBA consiste dos seguintes passos:

Passo 1: o usuário define o tipo XML do elemento primário da visão. XVBA oferece uma interface gráfica simples que permite a definição de tipos XML.

Passo 2: o usuário seleciona, de uma lista de tabelas e visões relacionais do esquema do banco, a tabela ou visão pivô, tal que existe um mapeamento 1-1 entre as tuplas de tabela/visão pivô e os elementos da visão.

Passo 3: o usuário especifica as assertivas de correspondência que definem o relacionamento entre os elementos e atributos do tipo XML da visão e os atributos/caminhos da tabela pivô. Figura 2.18 mostra uma tela da interface gráfica de XVBA que suporta a definição de assertivas de correspondência da visão. O esquema do lado esquerdo é o esquema XML da visão Pedidos_XML, cujo elemento primário <*pedido*> tem tipo TPedido_XML, e cujo esquema da relação pivô é *PEDIDOS_REL* (esquema do lado direito) do esquema relacional PEDIDOS_BD na . É importante notar que, a partir da relação/visão pivô, o usuário pode navegar para qualquer tabela através das chaves estrangeiras ou inversas de chaves estrangeiras.

As assertivas de correspondência de Pedidos_XML são geradas: (1) relacionando elementos e atributos de TPedido_XML com atributos ou caminhos de *PEDIDOS_REL*; e (2) recursivamente, definindo as assertivas dos sub-elementos de TPedido_XML. Por exemplo, para definir a assertiva do elemento Item (φ_{11}:[TPedido_XML/Item] ≡ [*PEDIDOS_REL*/FK_2^{-1}]), o usuário seleciona o elemento Item no esquema da visão e seleciona a inversa de chave estrangeira FK_2^{-1} no esquema do banco. Em seguida, o usuário deve especificar as assertivas de correspondência dos elementos de TItem_XML com atributos ou caminhos de *ITENS_REL* (φ_{12}, φ_{13}, φ_{17} e φ_{18}). Finalmente, o usuário deve especificar as assertivas de correspondência dos elementos de TProduto_XML com atributos ou caminhos de *PRODUTO_REL* (φ_{14}, φ_{15} e φ_{16}).

Passo 4: XVBA gera automaticamente, baseado nas assertivas de correspondência, a consulta SQL/XML que constrói os elementos XML da visão a partir das tuplas relacionais. mostra a definição SQL/XML da visão Pedidos_XML.

A ferramenta também suporta a manutenção das visões XML. Baseada nas assertivas, XVBA identifica automaticamente as visões XML afetadas pelas modificações no esquema da base de dados. Para cada visão afetada, deve-se redefinir as assertivas que se tornaram inválidas pelas modificaões no esquema da base de dados; e, baseado nas novas assertivas, a ferramenta gera a nova consulta SQL/XML.

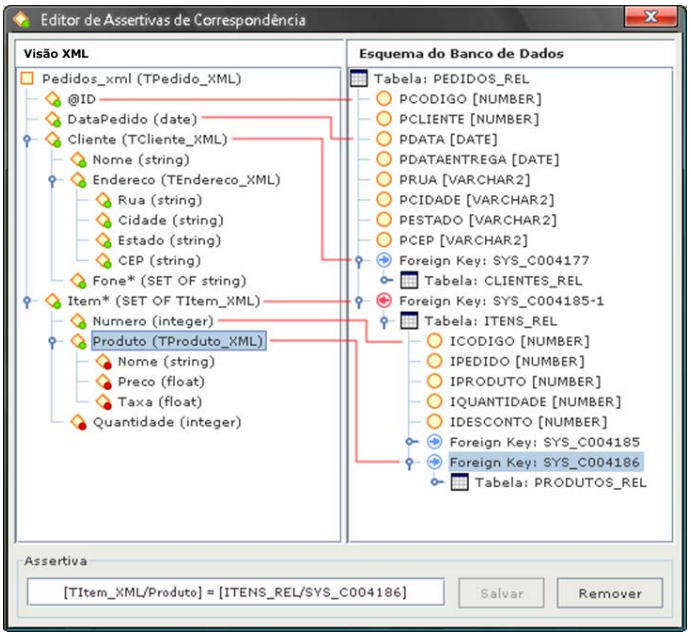

Figura 2.18. Editor de Assertivas de Correspondência

2.9 Conclusões

Neste capítulo, mostramos que podemos especificar completamente uma visão XML em termos de um esquema relacional usando assertivas de correspondência, no sentido que as assertivas definem um mapeamento de instâncias do esquema relacional em instâncias do esquema da visão XML.

Apresentamos um algoritmo que gera, baseado nas assertivas de correspondência da visão, a definição SQL/XML da visão. Além disso, mostramos que a consulta SQL/XML gerada pelo algoritmo representa corretamente o mapeamento definido pelas assertivas de correspondência da visão.

Apresentamos XVBA, uma ferramenta para facilitar a tarefa de criação de visões XML sob dados relacionais. A ferramenta oferece uma interface gráfica para apoiar a criação do tipo XML da visão e edição de suas assertivas de correspondência com o esquema base, e gera automaticamente a consulta SQL/XML, que realiza o mapeamento definido pelas assertivas da visão.

CAPÍTULO 3
Especificação e Geração de Visões XML sobre Múltiplas Bases de Dados

Neste capítulo, apresentamos um formalismo para especificação de visões XML sobre múltiplas fontes de dados (Visões de Integração de Dados). Esse formalismo é uma extensão do formalismo proposto no Capitulo 2. Além disso, propomos um algoritmo para gerar, a partir das assertivas de uma visão de integração de dados, o plano de materialização da visão, o qual computa o estado da visão a partir dos estados das fontes locais.

O capítulo está organizado da seguinte forma: A Seção 3.1 discute visões XML sobre múltiplas bases de dados; A Seção 3.2 apresenta uma extensão do nosso formalismo de mapeamento para especificação de visões de integração de dados; A Seção 3.3 discute como especificar visões de integração de dados usando assertivas de correspondência; A Seção 3.4 apresenta o algoritmo que gera automaticamente o plano de materialização da visão a partir das assertivas de correspondência; A Seção 3.5 apresenta as conclusões.

3.1 Introdução

Atualmente, grande parte das aplicações *Web* integra dados de fontes distintas e distribuídas. Em geral, o conteúdo dessas aplicações é modelado como um conjunto de visões XML definidas sobre múltiplas fontes de dados (Visões XML de Integração de Dados), de modo que a materialização da visão requer a integração dos dados das várias fontes.

Definir visões XML que integram dados é uma tarefa complexa, uma vez que devem ser especificadas quais fontes serão selecionadas e como mapear os dados das fontes em dados da visão. A heterogeneidade das fontes torna ainda mais complexo o processo de definição da visão, uma vez que os dados das fontes devem estar em formatos e esquemas comuns para que a integração possa ser realizada. Além disso, não é possível definir uma única consulta que mapeie o estado das fontes no estado

correspondente da visão. Desta forma, para materializar a visão, primeiramente são realizadas consultas nas fontes locais e em seguida os resultados são integrados através de operações de integração.

O plano de materialização de uma visão de integração é definido através de um *workflow* que, quando executado, computa o estado da visão a partir dos estados das fontes locais. Neste trabalho, o plano de materialização é representado por um grafo direcionado, denominado *Grafo de Materialização*. Em um grafo de materialização os nós representam processos e as arestas representam um fluxo de dados XML entre os processos. O resultado de cada processo é uma coleção XML a qual é repassada como parâmetro de entrada para o processo que se liga a este por uma seta.

No exemplo a seguir, apresentamos uma visão XML de integração de dados sobre bases relacionais. Considere a visão XML Autores_V, cujo tipo TAutor_V é mostrado na Figura 3.1. Considere as fontes de dados $AUTORES\_BD_1$ e $AUTORES\_BD_2$, cujos esquemas são mostrados nas Figuras 3.2 e 3.3 e considere os estados dos esquemas $AUTORES\_BD_1$ e $AUTORES\_BD_2$ mostradas nas Figuras 3.4 e 3.5, respectivamente. Para estes estados, o valor (estado) correspondente da visão é a coleção XML mostrada na Figura 3.6. Esta coleção contém um conjunto de elementos <*autor*>, obtidos da integração dos autores em $AUTORES\_BD_1$ e $AUTORES\_BD_2$.

Figura 3.7 mostra o plano de materialização da visão Autores_V. O *workflow* do plano é composto de três processos: PO, PP[$AUTORES\_REL_1$] e PP[$AUTORES\_REL_2$]. O processo PP[$AUTORES\_REL_1$] realiza a consulta Q_1 (Figura 3.8) sobre a relação $AUTORES\_REL_1$ da Figura 3.4 e retorna uma coleção XML de elementos <*autor*> cujo estado é mostrado na Figura 3.10. De forma semelhante, o processo PP[$AUTORES\_REL_2$] realiza a consulta Q_2 (Figura 3.9) sobre a relação $AUTORES\_REL_2$ da Figura 3.5 e retorna uma coleção XML de elementos <*autor*> cujo estado é mostrado na Figura 3.11.

Os processos PP[$AUTORES\_REL_1$] e PP[$AUTORES\_REL_2$] são executados em paralelo e seus resultados são entradas para o processo PO que então realiza a operação de *outer union* entre as coleções resultantes. Note que as coleções resultantes das consultas Q_1 e Q_2 têm estrutura compatível com o tipo da visão, a qual é condição necessária para a realização da operação de *outer union*.

Neste capítulo, apresentamos um formalismo para a especificação de visões XML virtuais sobre múltiplas fontes de dados. Esse formalismo é uma extensão ao formalismo proposto no Capítulo 2. Apresentamos também um conjunto de operações de integração necessárias para a integração dos dados.

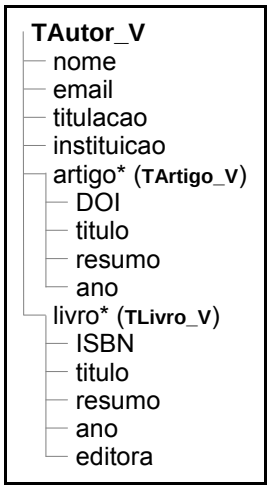

Figura 3.1. Tipo TAutor_V

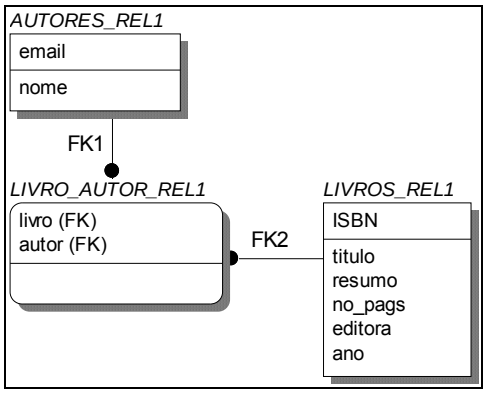

Figura 3.2. Esquema AUTORES_BD$_1$

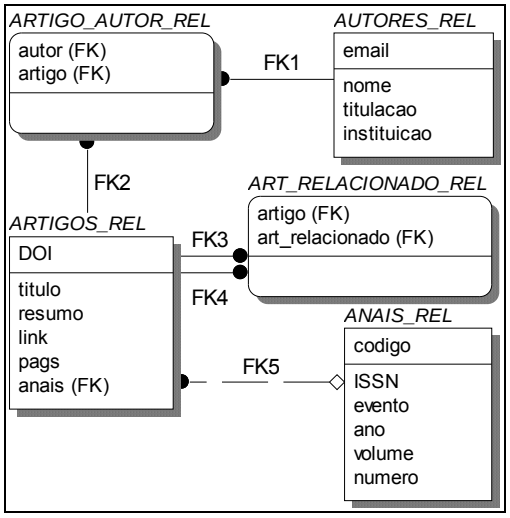

Figura 3.3. Esquema relacional AUTORES_BD$_2$

AUTORES_REL$_1$

EMAIL	NOME
andrew.eisenberg@us.ibm.com	Andrew Eisenberg
stephen.buxton@oracle.com	Stephen Buxton
lenzerini@dis.uniroma1.it	Maurizio Lenzerini
jim.melton@acm.org	Jim Melton

AUTOR_LIVRO_REL$_1$

LIVRO	AUTOR
1558605622	andrew.eisenberg@us.ibm.com
1558607110	stephen.buxton@oracle.com
3540420894	lenzerini@dis.uniroma1.it
1558605622	jim.melton@acm.org
1558607110	jim.melton@acm.org

LIVROS_REL$_1$

ISBN	TITULO	RESUMO	NO_PAGS	EDITORA	ANO
1558605622	Understanding SQL and Java Together	With the growth of Java and the rise...	512	Morgan Kaufmann	2000
1558607110	Querying XML: XQuery, XPath, and SQL/XML in Context	In a remarkably short period, XML...	848	Elsevier Books	2006
3540420894	Fundamentals of Data Warehouses	Data warehouses have captured...	235	Springer-Verlag	2001

Figura 3.4. Estado de AUTORES_BD$_1$

AUTORES_REL$_2$

EMAIL	NOME	TITULACAO	INSTITUICAO
jim.melton@acm.org	Jim Melton	Ph.D.	ORACLE
lenzerini@dis.uniroma1.it	Maurizio Lenzerini	Ph.D.	Università di Roma
andrew.eisenberg@us.ibm.com	Andrew Eisenberg	Ph.D.	IBM
lucian@almaden.ibm.com	Lucian Popa	Ph.D.	IBM

ARTIGOS_REL$_2$

DOI	TITULO	RESUMO	LINK	PAGS	ANAIS
10.1145/565117.565141	SQL/XML is making good progress	Not very long ago, we discussed…	www.acm.org/...	101-108	SIGREC
10.1145/543613.543644	Data integration: a theoretical perspective	Data integration is the problem of…	www.acm.org/...	233-246	SPDS
10.1O16/j.tcs.2004.10.033	Data exchange:semantics and query answering	Data exchange is the problem of…	www.acm.org/...	89-124	TCS

AUTOR_ARTIGO_REL$_2$

ARTIGO	AUTOR
10.1145/543613.543644	lenzerini@dis.uniroma1.it
10.1145/565117.565141	andrew.eisenberg@us.ibm.com
10.1145/565117.565141	jim.melton@acm.org
10.1O16/j.tcs.2004.10.033	lucian@almaden.ibm.com

ANAIS_REL$_2$

CODIGO	ISSN	ANO	VOLUME	EDICAO
SIGREC	0163-5808	2003	31	2
SPDS	1-58113-507-6	2002	NULL	NULL
TCS	0304-3975	2005	336	1

Figura 3.5. Estado de AUTORES_BD$_2$

```
<autor>
 <nome>Andrew Eisenberg</nome>
 <email>andrew.eisenberg@us.ibm.com</email>
 <titulacao>Ph.D.</titulacao>
 <instituicao>IBM</instituicao>
 <artigo>
   <DOI>10.1145/565117.565141</DOI>
   <titulo>SQL/XML is making good progress</titulo>
   <resumo>Not very long ago, we discussed...</resumo>
   <ano2003</ano>
 </artigo>
 <livro>
   <ISBN>1558605622</ISBN>
   <titulo>Understanding SQL and Java Together</titulo>
   <resumo>With the growth of Java and the rise...</resumo>
   <ano>2000</ano>
   <editora>Morgan Kaufmann</editora>
 </livro>
</autor>
<autor>
 <nome>Stephen Buxton</nome>
 <email>stephen.buxton@oracle.com</email>
 <livro>
   <ISBN>1558607110</ISBN>
   <titulo>Querying XML: XQuery, XPath, and SQL/XML in
Context</titulo>
   <resumo>In a remarkably short period, XML...</resumo>
   <ano>2006</ano>
   <editora>Elsevier Books</editora>
 </livro>
</autor>
<autor>
 <nome>Maurizio Lenzerini</nome>
 <email>lenzerini@dis.uniroma1.it</email>
 <titulacao>Ph.D.</titulacao>
 <instituicao>Università di Roma</instituicao>
 <artigo>
   <DOI>10.1145/543613.543644</DOI>
   <titulo>Data integration: a theoretical perspective</titulo>
   <resumo>Data integration is the problem of...</resumo>
   <ano>2002</ano>
 </artigo>
 <livro>
   <ISBN>3540420894</ISBN>
   <titulo>Fundamentals of Data Warehouses</titulo>
   <resumo>Data warehouses have captured...</resumo>
   <ano>2001</ano>
   <editora>Springer-Verlag</editora>
 </livro>
</autor>
<autor>
 <nome>Jim Melton</nome>
 <email>jim.melton@acm.org</email>
 <titulacao>Ph.D.</titulacao>
 <instituicao>ORACLE</instituicao>
 <artigo>
   <DOI>10.1145/565117.565141</DOI>
   <titulo>SQL/XML is making good progress</titulo>
   <resumo>Not very long ago, we discussed...</resumo>
   <ano>2003</ano>
 </artigo>
 <livro>
   <ISBN>1558605622</ISBN>
   <titulo>Understanding SQL and Java Together</titulo>
   <resumo>With the growth of Java and the rise...</resumo>
   <ano>2000</ano>
   <editora>Morgan Kaufmann</editora>
 </livro>
 <livro>
   <ISBN>1558607110</ISBN>
   <titulo>Querying XML: XQuery, XPath, and SQL/XML in
Context</titulo>
   <resumo>In a remarkably short period, XML...</resumo>
   <ano>2006</ano>
   <editora>Elsevier Books</editora>
 </livro>
</autor>
<autor>
 <nome>Lucian Popa</nome>
 <email>lucian@almaden.ibm.com</email>
 <titulacao>Ph.D.</titulacao>
 <instituicao>IBM</instituicao>
 <artigo>
   <DOI>10.1O16/j.tcs.2004.10.033</DOI>
   <titulo>Data exchange: semantics and query answering</titulo>
   <resumo>Data exchange is the problem of...</resumo>
   <ano>2005</ano>
 </artigo>
</autor>
```

Figura 3.6. Estado da visão Autores_V

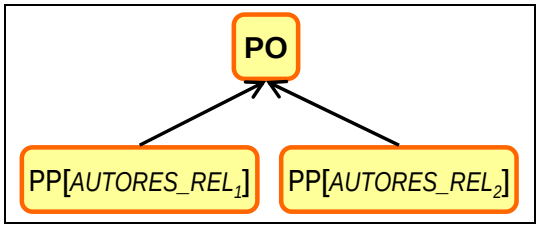

Figura 3.7. Plano de Materialização de Autores_V

```
SELECT XMLElement("autor",
  XMLForest("nome", AU.nome),
  XMLForest("email", AU.email),
  (SELECT XMLAgg( XMLElement("livro",
     XMLForest("ISBN", LI.ISBN),
     XMLForest("titulo", LI.titulo),
     XMLForest("resumo", LI.resumo),
     XMLForest("ano", LI.ano),
     XMLForest("editora", LI.editora) ) )
   FROM LIVRO_AUTOR_REL1 LA, LIVROS_REL1 LI
   WHERE LA.livro = AU.email AND
         LA.livro = LI.ISBN) )
FROM AUTORES_REL1 AU
```

Figura 3.8. Consulta Q_1

```
SELECT XMLElement("autor",
  XMLForest("nome", AU.nome),
  XMLForest("email", AU.email),
  XMLForest("titulacao", AU.titulacao),
  XMLForest("instituicao", AU.instituicao),
  (SELECT XMLAgg( XMLElement("artigo",
     XMLForest("DOI", AR.DOI),
     XMLForest("titulo", AR.titulo),
     XMLForest("resumo", AR.resumo),
     XMLForest("ano",
        (SELECT AN.ano
         FROM ANAIS_REL AN
         WHERE AR.anais = AN.codigo)) ) )
   FROM ARTIGO_AUTOR_REL2 AA,
        ARTIGOS_REL2 AR
   WHERE AA.autor = AU.email AND
         AA.artigo = AR.DOI) )
FROM AUTORES_REL2 AU
```

Figura 3.9. Consulta Q_2

```
<autor>
 <nome>Andrew Eisenberg</nome>
 <email>andrew.eisenberg@us.ibm.com</email>
 <livro>
   <ISBN>1558605622</ISBN>
   <titulo>Understanding SQL and Java Together</titulo>
   <resumo>With the growth of Java and the rise...</resumo>
   <ano>2000</ano>
   <editora>Morgan Kaufmann</editora>
 </livro>
</autor>
<autor>
 <nome>Stephen Buxton</nome>
 <email>stephen.buxton@oracle.com</email>
 <livro>
   <ISBN>1558607110</ISBN>
   <titulo>Querying XML: XQuery, XPath, and SQL/XML in
Context</titulo>
   <resumo>In a remarkably short period, XML...</resumo>
   <ano>2006</ano>
   <editora>Elsevier Books</editora>
 </livro>
</autor>
```

```
<autor>
 <nome>Maurizio Lenzerini</nome>
 <email>lenzerini@dis.uniroma1.it</email>
 <livro>
   <ISBN>3540420894</ISBN>
   <titulo>Fundamentals of Data Warehouses</titulo>
   <resumo>Data warehouses have captured...</resumo>
   <ano>2001</ano>
   <editora>Springer-Verlag</editora>
 </livro>
</autor>
<autor>
 <nome>Jim Melton</nome>
 <email>jim.melton@acm.org</email>
 <livro>
   <ISBN>1558605622</ISBN>
   <titulo>Understanding SQL and Java Together</titulo>
   <resumo>With the growth of Java and the rise...</resumo>
   <ano>2000</ano>
   <editora>Morgan Kaufmann</editora>
 </livro>
 <livro>
   <ISBN>1558607110</ISBN>
   <titulo>Querying XML: XQuery, XPath, and SQL/XML in
Context</titulo>
   <resumo>In a remarkably short period, XML...</resumo>
   <ano>2006</ano>
   <editora>Elsevier Books</editora>
 </livro>
</autor>
```

Figura 3.10. Estado da coleção resultante da consulta Q_1

```
<autor>
 <nome>Jim Melton</nome>
 <email>jim.melton@acm.org</email>
 <titulacao>Ph.D.</titulacao>
 <instituicao>ORACLE</instituicao>
 <artigo>
   <DOI>10.1145/565117.565141</DOI>
   <titulo>SQL/XML is making good progress</titulo>
   <resumo>Not very long ago, we discussed...</resumo>
   <ano>2003</ano>
 </artigo>
</autor>
<autor>
 <nome>Maurizio Lenzerini</nome>
 <email>lenzerini@dis.uniroma1.it</email>
 <titulacao>Ph.D.</titulacao>
 <instituicao>Università di Roma</instituicao>
 <artigo>
   <DOI>10.1145/543613.543644</DOI>
   <titulo>Data integration: a theoretical perspective</titulo>
   <resumo>Data integration is the problem of...</resumo>
   <ano>2002</ano>
 </artigo>
</autor>
```

```
<autor>
 <nome>Andrew Eisenberg</nome>
 <email>andrew.eisenberg@us.ibm.com</email>
 <titulacao>Ph.D.</titulacao>
 <instituicao>IBM</instituicao>
 <artigo>
   <DOI>10.1145/565117.565141</DOI>
   <titulo>SQL/XML is making good progress</titulo>
   <resumo>Not very long ago, we discussed...</resumo>
   <ano2003</ano>
 </artigo>
</autor>
<autor>
 <nome>Lucian Popa</nome>
 <email>lucian@almaden.ibm.com</email>
 <titulacao>Ph.D.</titulacao>
 <instituicao>IBM</instituicao>
 <artigo>
   <DOI>10.1O16/j.tcs.2004.10.033</DOI>
   <titulo>Data exchange: semantics and query
answering</titulo>
   <resumo>Data exchange is the problem of...</resumo>
   <ano>2005</ano>
 </artigo>
</autor>
```

Figura 3.11. Estado da coleção resultante da consulta Q_2

3.2 Operações de Integração de Dados

Nesta seção, apresentamos as quatro operações utilizadas para integração de dados XML: *fusão*, *outer union*, *left outer union* e *left outer join*.

Neste trabalho, uma operação de integração é realizada com base em um tipo XML, que é o tipo dos elementos resultantes da operação. Os tipos dos elementos dos operandos são tipos reduzidos desse tipo, conforme definido a seguir:

Sejam T e T' tipos XML. Dizemos que T' *é um tipo reduzido de* T, denotado por T'«T, sse para cada elemento (atributo) p de T', onde T'_p é o tipo de p:

- Se T'_p é de tipo simples, então p é um elemento (atributo) de T e tem tipo T'_p.
- Se T'_p é de tipo complexo, então p é um elemento (atributo) de T e tem tipo T_p, tal que T'_p é um tipo reduzido de T_p.

Informalmente, um tipo reduzido de outro é um tipo com número de propriedades ou sub-propriedades menor ou igual ao do outro tipo. Isso porquê após realizarmos uma operação de integração entre dois elementos, o elemento resultante não pode ser de um tipo com menos propriedades ou sub-propriedades que o dos tipos dos operandos.

No que se segue, dada uma coleção XML C, $C denota o estado de C.

3.2.1 Operação Fusão

A operação de fusão é utilizada para combinar (*merge*) dois elementos, que representam o mesmo elemento no mundo real, em um único elemento.

Suponha os elementos $\$t_1$ e $\$t_2$, instâncias dos tipos XML T_1 e T_2, respectivamente. Seja T um tipo XML tal que $T_1 \ll T$ e $T_2 \ll T$. A *fusão de* $\$t_1$ *e* $\$t_2$ *com base no tipo* T, denotada por $\$t_1 \overset{\infty}{}_{T} \$t_2$, é um elemento XML $\$t$ de tipo T, tal que para toda propriedade p de T, temos que:

- Se p tem ocorrência simples e tem tipo simples, então

  $\$t/p = \{ \$p \mid \$p = \$t_1/p$, onde $\$t_1/p \neq \emptyset$, ou

  $\$p = \$t_2/p$, onde $\$t_2/p \neq \emptyset\}$
- Se p tem ocorrência simples e tem tipo complexo T_P, então

  $\$t/p = \{ \$p \mid \$p = \$t_1/p$, onde $\$t_1/p \neq \emptyset$ e $\$t_2/p = \emptyset$, ou

  $\$p = \$t_2/p$, onde $\$t_1/p = \emptyset$ e $\$t_2/p \neq \emptyset$, ou

  $\$p = \$t_1/p \overset{\infty}{}_{T_P} \$t_2/p$, onde $\$t_1/p \neq \emptyset$ e $\$t_2/p \neq \emptyset$ }
- Se p tem ocorrência múltipla, então

  $\$t/p = (\$t_1/p \mathbin{\mathring{\cup}}_{T_P} \$t_2/p)$, onde T_P é o tipo de p. O operador $\mathring{\cup}$ (*outer union*) será definido em seguida.

Figura 3.12 mostra o algoritmo RealizaFusão que implementa a operação de fusão.

INPUT: instância t_1$ do tipo XML T_1, instância t_2$ do tipo XML T_2, tipo XML T tal que T_1«T e T_2«T, rótulo *r* para o elemento resultante da fusão
OUTPUT: instância $t de T, tal que $t = t_1$ $\overset{\infty}{}_{T}$ t_2$

```
$t := criaElemento( r );
Para cada propriedade p de T Faça
    Seja Tp o tipo de p;
    No Caso:
        Caso 1: Se p têm ocorrência simples e Tp é simples Então
            Se $t1/p ≠ ∅ Então adicionaFilho( $t1/p, $t );
            Se $t2/p ≠ ∅ Então adicionaFilho( $t1/p, $t );
        Caso 2: Se p têm ocorrência simples e Tp é complexo Então
            Se $t1/p ≠ ∅ e $t2/p = ∅ Então adicionaFilho( $t1/p, $t );
            Se $t1/p = ∅ e $t2/p ≠ ∅ Então adicionaFilho( $t2/p, $t );
            Se $t1/p ≠ ∅ e $t2/p ≠ ∅ Então adicionaFIlho( $t1/p ∞ $t2/p, $t );
        Caso 3: Se p têm ocorrência múltipla Então
            $P = RealizaOuterUnion( $t1/p, $t2/p, T, K);
            Para cada $p∈$P Faça
                adicionaFilho($p, $t);
            Fim Para;
    Fim Caso;
Fim Para;
Retorne $t;

NOTA:
- O método criaElemento( r ) cria um nó com rótulo r;
- O método adicionaFilho( $c, $p ) adiciona o nó $c como filho do nó $p;
```

Figura 3.12. Algoritmo RealizaFusão

Por exemplo, considere os elementos a_1$ e a_2$ (Figura 3.16), instâncias dos tipos XML TArtigo$_1$ e TArtigo$_2$ mostrados nas Figuras 3.13 e 3.14, respectivamente. Considere o tipo XML TArtigo mostrado na Figura 3.15. Note que TArtigo$_1$ e TArtigo$_2$ são compatíveis e que TArtigo$_1$«TArtigo e TArtigo$_2$«TArtigo. O resultado da fusão de a_1$ e a_2$ com relação à TArtigo (a_1$ $\overset{\infty}{}_{TArtigo}$ a_2$) é o elemento mostrado na Figura 3.16.

De acordo com o algoritmo, primeiramente são analisadas as propriedades de TArtigo: DOI, título, resumo, ano e autor. No caso das propriedades DOI, título, resumo e ano, como estas têm ocorrência simples e seus tipos são simples, então recai no Caso 1 do algoritmo (linhas 6-7). No caso da propriedade autor, como autor têm ocorrência múltipla, então recai no Caso 3 do algoritmo (linhas 13-15). A operação de *outer union*, realizada entre as coleções a_1$/autor e a_2$/autor retorna uma coleção que contém os elementos de a_1$/autor e a_2$/autor, sendo que é realizada a fusão com base no tipo TAutor dos elementos duplicados. Note que os elementos <*autor*> que representam o mesmo autor no mundo real aparecem somente uma vez no resultado.

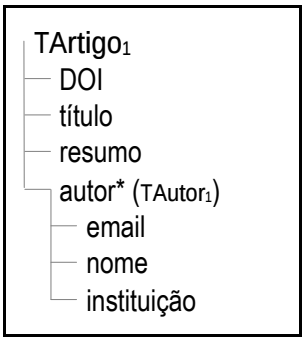

Figura 3.13. Tipo $TArtigo_1$

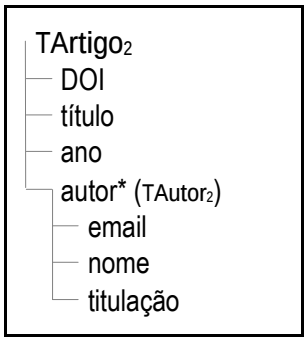

Figura 3.14. Tipo $TArtigo_2$

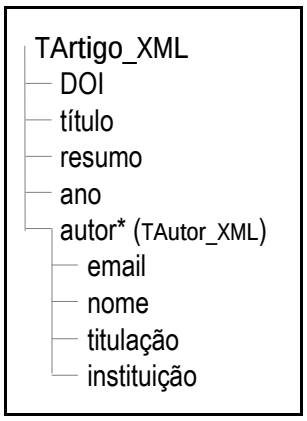

Figura 3.15. Tipo TArtigo

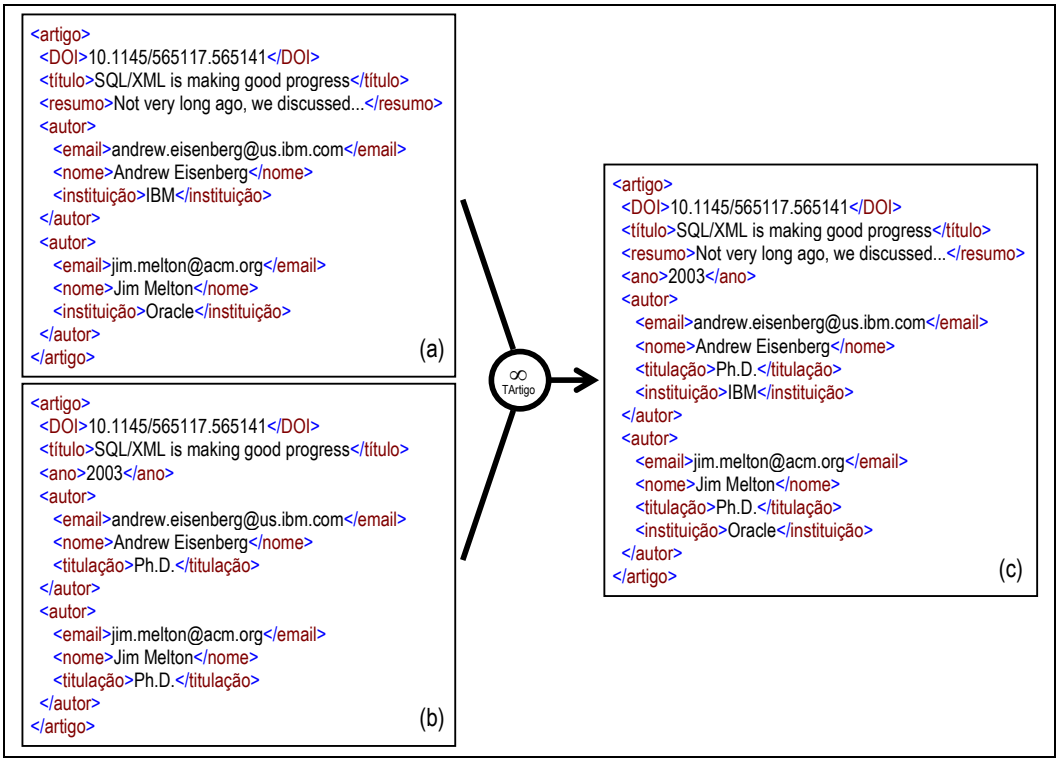

Figura 3.16. (a) Instância $\$a_1$ de TArtigo; (b) Instância $\$a_2$ de TArtigo; (c) Resultado de $\$a_1 \overset{\infty}{}_{TArtigo} \$a_2$

3.2.2 Operação *Outer Union*

O resultado de uma operação *outer union* entre duas coleções XML é uma coleção XML que inclui todos os elementos das coleções de entrada, sendo que é realizada a fusão dos elementos que representam o mesmo objeto do mundo real. Para realizar a *outer union* entre duas coleções XML C_1 e C_2, cujos esquemas são $C_1=<e_1, T_1>$ e $C_2=<e_2, T_2>$, estas devem ser *compatíveis para a outer union*, isto é:

(i) T_1 e T_2 são tipos compatíveis,

(ii) C_1 e C_2 têm uma chave $K=\{a_1, ..., a_n\}$ em comum, onde a_1, ..., a_n são propriedades comuns de T_1 e T_2, e

(iii) as coleções aninhadas em comum de C_1 e C_2 são compatíveis para a *outer union*.

Mais formalmente, sejam C_1 e C_2 coleções XML cujos esquemas são $C_1=<e_1, T_1>$ e $C_2=<e_2, T_2>$, respectivamente. Suponha que C_1 e C_2 sejam compatíveis para a *outer union* e que K é uma chave comum de C_1 e C_2. A *outer union de* C_1 *e* C_2 *com base no tipo* T, onde $T_1 « T$ e $T_2 « T$, denotada por $C_1 \mathring{\cup}_T C_2$, é uma coleção XML C de esquema $C=<e, T>$, cujo estado é dado por:

$$\$C = \{ \$s \mid \$s=\$s_1, \text{ onde } \$s_1 \in \$C_1 \text{ e } \neg\exists\, \$s_2 \in \$C_2 \text{ tal que } \$s_1[K]=\$s_2[K], \text{ ou}$$
$$\$s=\$s_2, \text{ onde } \neg\exists\, \$s_1 \in \$C_1 \text{ e } \$s_2 \in \$C_2 \text{ tal que } \$s_1[K]=\$s_2[K], \text{ ou}$$
$$\$s=\$s_1 \overset{\circ}{\cup}_T \$s_2, \text{ onde } \$s_1 \in \$C_1 \text{ e } \$s_2 \in \$C_2 \text{ tal que } \$s_1[K]=\$s_2[K] \}.$$

Figura 3.17 mostra o algoritmo RealizaOuterUnion que implementa a operação de *outer union*.

INPUT: coleção XML C_1 de esquema $<e, T_1>$, coleção XML C_2 de esquema $<e, T_2>$, tipo XML T tal que $T_1 « T$ e $T_2 « T$, chave $K=\{a_1, ..., a_n\}$ comum de C_1 e C_2
OUTPUT: coleção XML C de esquema $C=<e, T>$, tal que $C = C_1 \mathring{\cup}_T C_2$

```
Seja $C o estado da coleção XML C;
$C = ∅;
Para cada instância $c1∈$C1 Faça
   Se ¬∃ $c2∈$C2 tal que $c1[K] = $c2[K] Então
      $C := $C ∪ {$c1};
   Senão
      $c = RealizaFusão($c1, $c2, T, e);
      $C := $C ∪ {$c};
   Fim Se;
Fim Para;
Para cada instância $c2∈$C2 Faça
   Se ¬∃ $c1∈$C1 tal que $c1[K] = $c2[K] Então
      $C := $C ∪ {$c2};
   Fim Se;
Fim Para;
Retorne C;
```

Figura 3.17. Algoritmo RealizaOuterUnion

Por exemplo, considere as coleções XML $Artigos_1$ e $Artigos_2$ cujos estados são mostrados na Figura 3.18 e cujos esquemas são $<autor, TArtigo_1>$ e $<autor, TArtigo_2>$, os quais são compatíveis. Seja TArtigo_XML um tipo XML tal que $TArtigo_1$_XML « TArtigo_XML e $TArtigo_2$_XML « TArtigo_XML, e suponha $K=\{DOI\}$ uma chave comum de $Artigos_1$ e $Artigos_2$. O resultado da *outer union* entre $Artigos_1$ e $Artigos_2$ com base em TArtigo ($Artigos_1 \mathring{\cup}_{TArtigo} Artigos_2$) é mostrado na Figura 3.18(c).

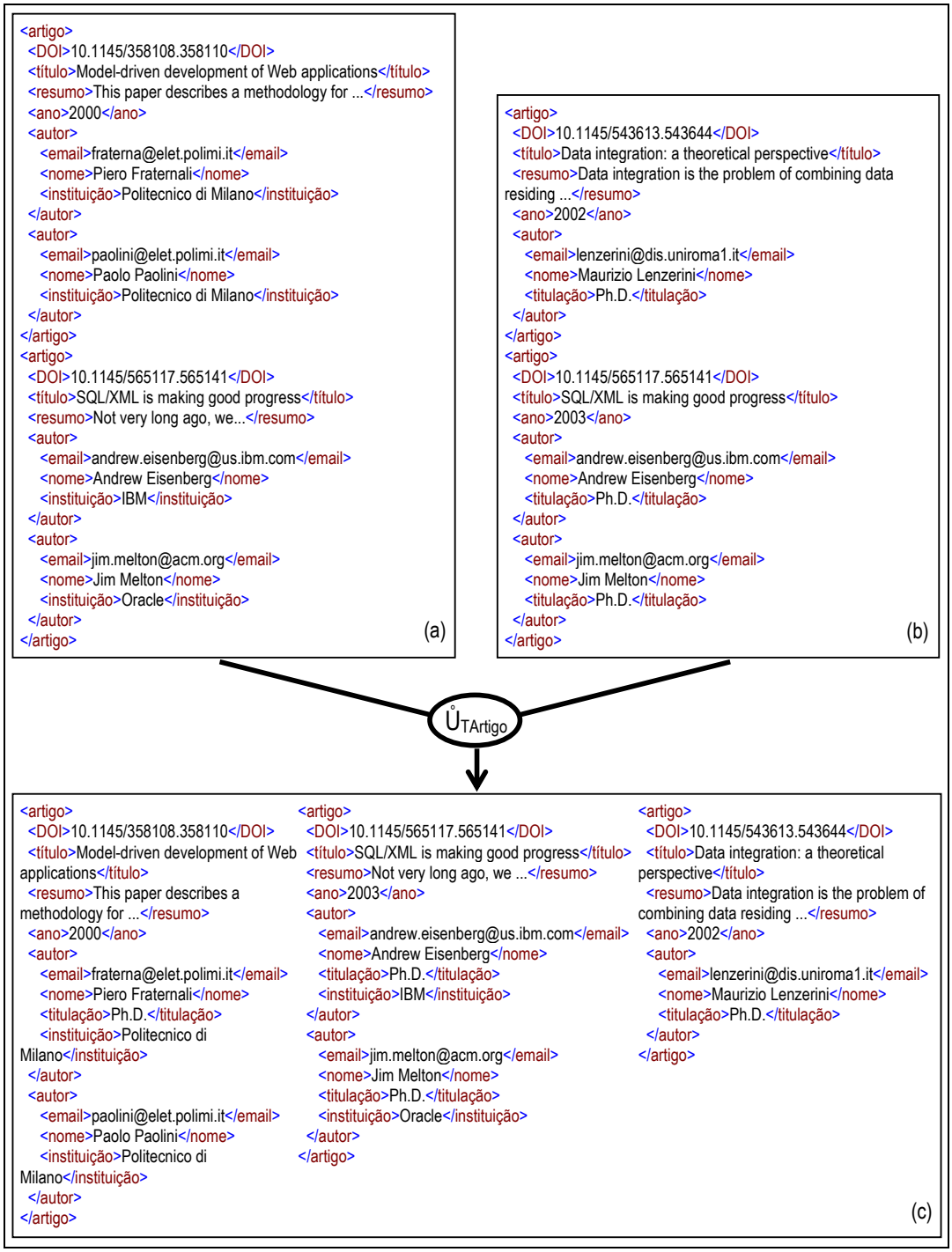

Figura 3.18. (a) Estado da coleção Artigos$_1$; (b) Estado da coleção Artigos$_2$; (c) Resultado de Artigos$_1$ Û$_{TArtigo}$ Artigos$_2$

De acordo com o algoritmo, primeiro são analisados os elementos da coleção Artigos$_1$. Deste modo, para o elemento a_{11} = $Artigos$_1$[DOI=10.1145/358108.358110], como não existe $a∈$Artigos$_2$ tal que $a/DOI = a_{11}$/DOI, então a_{11}$ é incluído no resultado da operação. Para o elemento a_{12} = $Artigos$_1$[DOI=10.1145/565117.565141],

como existe $a∈$Artigos$_2$ tal que $a/DOI = a_{1_2}$/DOI), então o elemento resultante de a_{1_2}$ ∞ $a é incluído no resultado da operação.

Em seguida, o algoritmo analisa os elementos de Artigos$_2$_XML. Para o elemento a_{2_1}$ = $Artigos$_2$ [DOI=10.1145/543613.543644], como não existe $a∈$Artigos$_1$ tal que $a/DOI = a_{2_1}$/DOI, então a_{2_1}$ é incluído no resultado da operação. Para o elemento a_{2_2}$ = $Artigos$_2$ [DOI=10.1145/565117.565141], apesar de existir $a∈$Artigos$_1$ tal que $a/DOI = a_{2_2}$/DOI, a_{2_2}$ é descartado e não entra no resultado final da operação. Note que a_{2_2}$ já foi considerado para o resultado final da operação na primeira iteração do algoritmo.

3.2.3 Operação *Left Outer Union*

O resultado de uma operação *left outer union* entre duas coleções XML, A e B, é uma coleção XML C que inclui todos os elementos de A sendo que, no caso dos elementos comuns em A e B, é realizada a fusão dos dois elementos. Para realizar a *left outer union* entre duas coleções XML C_1 e C_2, cujos esquemas são C_1=<e_1, T_1> e C_2=<e_2, T_2>, estas devem ser *compatíveis para a left outer union*, isto é:

(iv) T_1 e T_2 são tipos compatíveis,

(v) C_1 e C_2 têm uma chave K={a_1, ..., a_n} em comum, onde a_1, ..., a_n são propriedades comuns de T_1 e T_2, e

(vi) as coleções aninhadas em comum de C_1 e C_2 são compatíveis para a *outer union*.

Mais formalmente, sejam C_1 e C_2 coleções XML de esquemas C_1=<e_1, T_1> e C_2=<e_2, T_2>, respectivamente. Suponha que C_1 e C_2 sejam compatíveis e que K é uma chave comum de C_1 e C_2. O resultado da *left outer union entre* C_1 *e* C_2 *com base no tipo* T, onde T_1«T e T_2«T, denotado por C_1 ⟕$_T$ C_2, é uma coleção XML C de esquema C=<e, T>, cujo estado é dado por:

$C = { $s | $s=$s$_1$, onde s_1$∈$C$_1$ e ¬∃ s_2$∈$C$_2$ tal que s_1$[K]=$s$_2$[K], ou

$s=$s$_1$ ⟕$_T$ s_2$, onde s_1$∈C_1$ e s_2$∈$C$_2$ tal que s_1$[K]=$s$_2$[K] }.

Figura 3.19 mostra o algoritmo RealizaLeftOuterUnion que implementa a operação de *left outer union*.

INPUT: coleção XML C_1 de esquema <e, T_1>, coleção XML C_2 de esquema <e, T_2>, tipo XML T tal que T_1«T e T_2«T, chave K={a_1, ..., a_n} comum de C_1 e C_2
OUTPUT: coleção XML C de esquema C=<e, T>, tal que C = C_1 ⟕$_T$ C_2

```
Seja $C o estado da coleção XML C;
$C = ∅;
Para cada instância $c1∈$C1 Faça
   Se ¬∃ $c2∈$C2 tal que $c1[K] = $c2[K] Então
      $C := $C ∪ {$c1};
   Senão
      $c = RealizaFusão($c1, $c2, T, e);
      $C := $C ∪ {$c};
   Fim Se;
Fim Para;
Retorne C;
```

Figura 3.19. Algoritmo RealizaLeftOuterUnion

Por exemplo, considere as coleções XML $Artigos_1$ e $Artigos_2$ cujos estados são mostrados na Figura 3.20, e cujo esquemas são <autor, $TArtigo_1$> e <autor, $TArtigo_2$>, os quais são compatíveis. Seja TArtigo_XML um tipo XML tal que $TArtigo_1$_XML«TArtigo_XML e $TArtigo_2$_XML«TArtigo_XML, e suponha K={DOI} uma chave comum de $Artigos_1$ e $Artigos_2$. O resultado da *left outer union* entre $Artigos_1$ e $Artigos_2$_XML com base em TArtigo ($Artigos_1$ ⟕$_{TArtigo}$ $Artigos_2$) é mostrado na Figura 3.20(c).

De acordo com o algoritmo, são analisados os elementos pertencentes à coleção $Artigos_1$. Deste modo, para o elemento $\$a_{1_1}$ = $Artigos_1$[DOI= 10.1145/358108.358110], como não existe $\$a \in Artigos_2$ tal que $\$a/DOI = \$a_{1_1}/DOI$, então $\$a_{1_1}$ é inserido no resultado da operação. Para o elemento $\$a_{1_2}$ = $Artigos_1$[DOI=10.1145/565117.565141], como existe $\$a \in Artigos_2$ tal que $\$a/DOI = \$a_{1_2}/DOI$, então o elemento resultante de $\$a_{1_2} \infty \$a$ é inserido no resultado da operação.

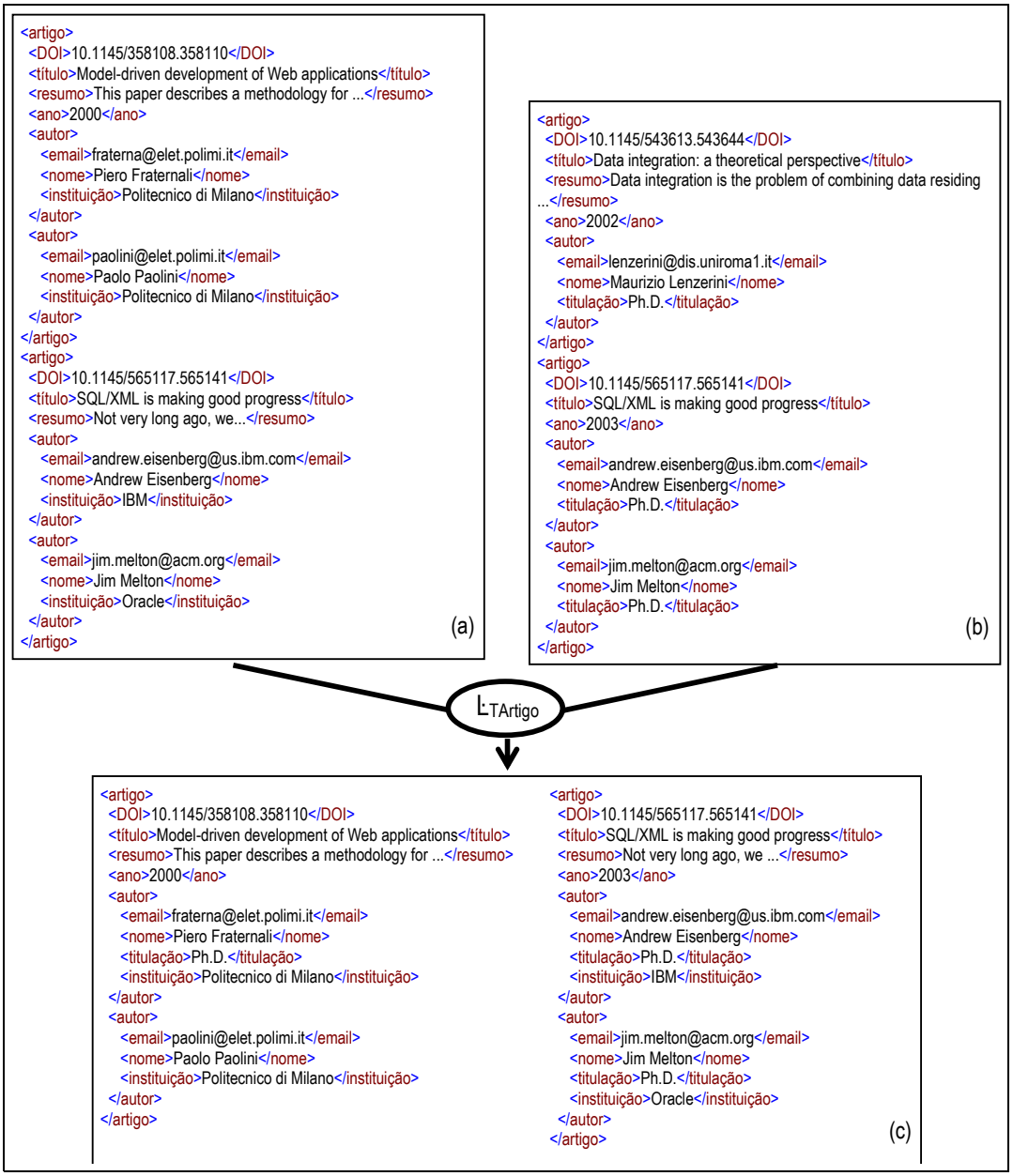

Figura 3.20. (a) Estado da coleção XML $Artigos_1$; (b) Estado da coleção XML $Artigos_2$; (c) Resultado de $Artigos_1$ $\leftthreetimes_{TArtigo}$ $Artigos_2$

3.2.4 Operação *Left Outer Join*

O resultado de uma operação *left outer join* entre duas coleções XML, A e B, com base em um caminho δ de A, é uma coleção XML C que inclui todos os elementos de A, sendo que para cada elemento $a de A é realizada a *left outer union de* $a/δ com B ($a/δ ⟕ B). Note que, para realizar a *left outer join* entre duas coleções XML A e B com base no caminho δ de A, as coleções A(δ) e B devem ser compatíveis para a *left outer union*.

Figura 3.21 mostra o algoritmo RealizaLeftOuterJoin que implementa a operação de *left outer join*.

```
INPUT: coleção XML C1 de esquema <e1, T1>, coleção XML C2 de esquema <e2, T2>, caminho δ de T1,
tipo XML T tal que Tδ«T e T2«T, chave K={a1, ..., an} comum de C1(δ) e C2
OUTPUT: coleção XML C de esquema C = <e1, T1>, tal que C = C1 Ĵ(T,δ) C2
```

```
Para cada instância $c1∈$C1 Faça
    Seja $E = LeftOuterUnion($c1/δ, C2);
    Para cada instância $e∈c1/δ Faça
        removeFilho( $e, $c1 );
    Fim Para;
    Para cada instância $e∈$E Faça
        adicionaFilho( $e, $c1/δ );
    Fim Para;
Fim Para;
Retorne C1;

NOTA:
- O método adicionaFilho( $c, $p ) adiciona o nó $c como filho do nó $p;
- O método removeFilho( $c, $p ) remove o nó filho $c do nó $p;
```

Figura 3.21. Algoritmo RealizaLeftOuterJoin

Por exemplo, considere as coleções XML Livros e Autores cujos estados são mostrados na Figura 3.22. O resultado da *left outer join* entre Livros e Autores com relação à TAutor com base no caminho autor (Livros $\hat{J}_{(TAutor,autor)}$ Autores) é mostrado na Figura 3.22. Intuitivamente, o algoritmo tem como resultado a coleção Livros, onde, para cada elemento $l∈Livros, substitui-se $l/autor pelo resultado da *left outer union* $l/autor $⟕_{TAutor,autor}$ Autores.

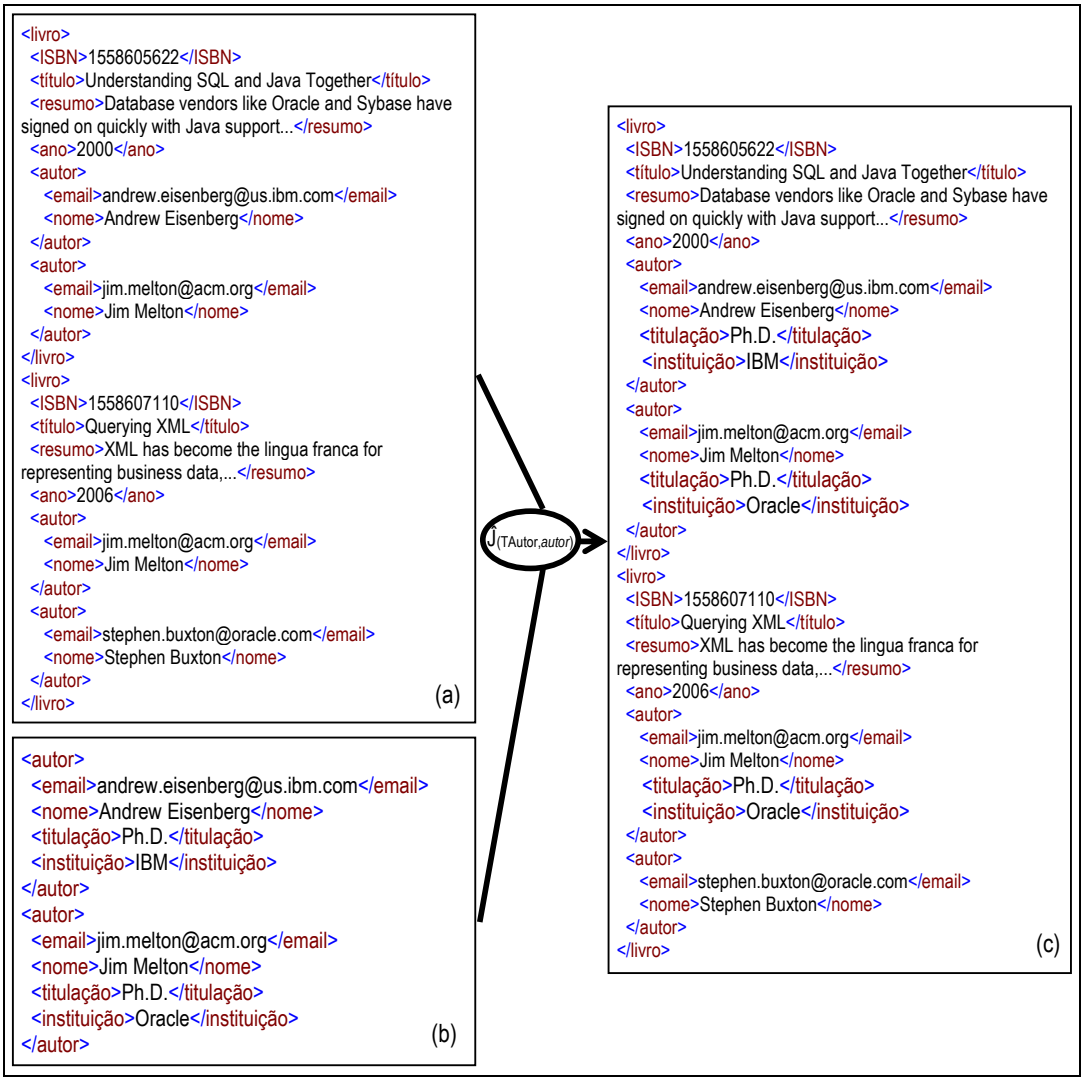

Figura 3.22. (a) Estado da coleção XML Livros; (b) Estado da coleção XML Autores; (c) Resultado de Livros $\hat{J}_{(TAutor,autor)}$ Autores

3.3 Assertivas de Correspondência de Caminhos de Tipos XML

Nas visões de integração, os elementos da visão são obtidos da integração de dados oriundos de várias bases. Desta forma, o formalismo de mapeamento utilizado para definir a visão deve permitir que o esquema da visão seja especificado em termos de mais de um esquema base.

Nesta seção, apresentamos uma extensão do formalismo das assertivas de correspondência, apresentado no Capítulo 2, para definir mapeamentos de esquema de visões XML sobre múltiplas bases de dados. O formalismo estendido permite

especificar como os elementos da visão de integração de dados são obtidos a partir dos elementos das várias coleções bases.

Definição 3.1: Sejam T e T' tipos XML. Suponha que e é um elemento/atributo de T e que δ é um caminho possivelmente nulo (δ=NULL) ou não definido (δ=∅) de T'. Uma *Assertiva de Correspondência de Caminho* (ACC) é uma expressão da forma:

- [T/e] ≡ [T'/δ] tal que e é compatível com δ; ou
- [T/e] ⊃ [T'/δ] tal que e é compatível com δ; ou
- [T/e] ≡ [T'/δ] ⊣⊤ [T'/δ/ℓ], onde ℓ é uma ligação da coleção aninhada no contexto T'/δ.

Assertivas da forma [T/e] ≡ [T'/δ] ⊣⊤ [T'/δ/ℓ] são denominadas *Assertivas de Junção*. □

Definição 3.2: Seja $\mathcal{A}$ um conjunto de assertivas de correspondência de caminho. Dizemos que $\mathcal{A}$ *especifica completamente* T *em termos de* T' sse:

(i) Para cada elemento e de T, existe uma única ACC em $\mathcal{A}$, chamada *a ACC para* e *em* $\mathcal{A}$.

(ii) Para cada assertiva em $\mathcal{A}$ da forma [T/e] ≡ [T'/δ] ou [T/e] ⊃ [T'/δ], onde e tem tipo complexo T_e, então $\mathcal{A}$ especifica completamente T_e em termos de T_δ, onde T_δ é o tipo de δ.

(iii) Para cada assertiva em $\mathcal{A}$ da forma [T/e] ≡ [T'/δ] ⊣⊤ [T'/δ/ℓ], onde e tem tipo complexo T_e, então:
   - $\mathcal{A}$ especifica completamente T_e em termos de T_δ; e
   - $\mathcal{A}$ especifica completamente T_e em termos de T'. □

Definição 3.3: Seja $\mathcal{A}$ um conjunto de assertivas de correspondência de caminho tal que $\mathcal{A}$ especifica completamente T em termos de T'.

(i) Suponha T_1 e T_2 tipos XML simples. Sejam S_1 e S_2 conjuntos de elementos XML de tipos T_1 e T_2, respectivamente. Dizemos que $S_1 \equiv_{\mathcal{A}} S_2$ sse, para todo $\$t_1$, $\$t_1 \in S_1$ sse existe $\$t_2 \in S_2$ tal que $\$t_1$/text() = $\$t_2$/text().

(ii) Suponha T_1 e T_2 tipos XML complexos.Sejam S_1 e S_2 conjuntos de elementos XML de tipos T_1 e T_2, respectivamente. Dizemos que $S_1 \equiv_{\mathcal{A}} S_2$ sse, para todo $\$t_1$, $\$t_1 \in S_1$ sse existe $\$t_2 \in S_2$ tal que $\$t_1 \equiv_{\mathcal{A}} \$t_2$.

(iii) Sejam S_1 e S_2 conjuntos de elementos XML instâncias dos tipos XML simples T_1 e T_2, respectivamente. Dizemos que $S_1 \supset_{\mathcal{A}} S_2$ sse, para todo $\$t_2$, se existe $\$t_2 \in S_2$, então $\exists\ \$t_1 \in S_1$, tal que $\$t_1/text() = \$t_2/text()$.

(iv) Sejam S_1 e S_2 conjuntos de elementos XML instâncias dos tipos XML complexos T_1 e T_2, respectivamente. Dizemos que $S_1 \supset_{\mathcal{A}} S_2$ sse, para todo $\$t_2$, se existe $\$t_2 \in S_2$, então $\exists\ \$t_1 \in S_1$, tal que $\$t_1 \equiv_{\mathcal{A}} \$t_2$.

(v) Seja $\$t$ uma instância de T e seja $\$t'$ uma instância de T'. Dizemos que $\$t \equiv_{\mathcal{A}} \$t'$ sse, para cada elemento/atributo e de T:

- se $[T/e] \equiv [T'/\delta]$ é a ACC para e em $\mathcal{A}$, então $\$t/e \equiv_{\mathcal{A}} \$t'/\delta$, onde $\delta \neq \varnothing$ e $\$t/e \neq \varnothing$;
- se $[T/e] \supset [T'/\delta]$ é a ACC para e em $\mathcal{A}$, então $\$t/e \supset_{\mathcal{A}} \$t'/\delta$ e $\$t/e \neq \varnothing$;
- se $[T/e] \equiv [T'/\delta] \llcorner_T [T'/\delta/\ell]$ é a ACC para e em $\mathcal{A}$, então $\$t/e \equiv_{\mathcal{A}} \$t'/\delta \llcorner_T \$t'/\delta/\ell$, onde $\delta \neq \varnothing$.

Se $\$t \equiv_{\mathcal{A}} \$t'$, dizemos que $\$t$ é *semanticamente equivalente a* $\$t'$ *como especificado por* $\mathcal{A}$.

□

Definição 3.4: Seja $\mathcal{A}$ um conjunto de assertivas de correspondência de caminho que especifica completamente T em termos de T_1 e … e T_n.

- Denotamos por $\mathcal{A}_{[T\&Ti]}$ o conjunto de assertivas de $\mathcal{A}$ que especificam completamente T em termos de T_i, $1 \leq i \leq n$.
- Denotamos por $\mathcal{A}^*_{[T\&Ti]}$ o conjunto de assertivas construído da seguinte forma: para toda assertiva $\varphi \in \mathcal{A}_{[T\&Ti]}$, temos que

  Se φ for da forma $[T/e] \equiv [T'/\delta]$ ou $[T/e] \supset [T'/\delta]$ então $\varphi \in \mathcal{A}^*_{[T\&Ti]}$.

  Se φ for da forma $[T/e] \equiv [T'/\delta] \llcorner_T [T'/\delta/\ell]$, então $\varphi' \in \mathcal{A}^*_{[T\&Ti]}$, onde $\varphi' = [T/e] \equiv [T'/\delta]$. □

3.4 Usando Assertivas de Correspondência para Especificar Visões XML de Integração de Dados

Propomos especificar uma Visão XML de Integração de Dados com o auxílio de um conjunto de assertivas de correspondência, que especificam de forma axiomática como os elementos da visão são sintetizados a partir dos elementos das fontes locais. Uma *Visão XML de Integração de Dados* é uma tripla $V = \langle S, \psi, \mathcal{A} \rangle$, onde:

(i) $S = \langle e, T \rangle$ é o esquema da visão;

(ii) ψ é uma assertiva de correspondência global da forma:

$V \equiv ((C_1[exp_1] \cup ... \cup C_k[exp_k]) \overset{\circ}{\cup}_T C_{k+1}[exp_{k+1}] \overset{\circ}{\cup}_T ... \overset{\circ}{\cup}_T C_{k+m}[exp_{k+m}]) \leftarrow_T C_{k+m+1} \leftarrow_T ... \leftarrow_T C_{k+m+n})$

onde C_i é uma coleção base, a qual pode ser uma tabela relacional, tabela de objetos ou coleção XML, e exp_i é uma expressão predicativa [45] da forma:

$$exp_i = (\delta_1 \theta_1 '\nu_1' \theta_2 ... \theta_{j-1} \delta_j \theta_j '\nu_j'),$$

onde $\delta_1, ..., \delta_j$ são caminhos de C_i, $\theta_1, ..., \theta_j$ são operadores booleanos e $\nu_1, ..., \nu_j$ são valores escalares. Dizemos que $C_1, \ldots, C_{k+m+n}$ são as *Coleções Bases* de V.

(iii) $\mathcal{A} = \bigcup_{i=1}^{k+m+n} \mathcal{A}_i$, onde $\mathcal{A}_i$ é um conjunto de assertivas de correspondência que especifica completamente T em termos de T_i (o tipo dos elementos de C_i), $1 \le i \le k+m+n$.

As assertivas de correspondência de V definem um mapeamento funcional dos estados correntes das coleções base para o estado da visão. Seja $\$C_1, \ldots, \$C_{k+m+n}$ os estados de $C_1, \ldots, C_{k+m+n}$ para o estado σ. O estado de V em σ, denotado por $\sigma(V)$, é dado por:

$$\sigma(V) = ((C'_1 \cup ... \cup C'_k) \overset{\circ}{\cup}_T C'_{k+1} \overset{\circ}{\cup}_T ... \overset{\circ}{\cup}_T C'_{k+m}) \leftarrow_T C'_{k+m+1} \leftarrow_T ... \leftarrow_T C'_{k+m+n}),$$

onde C'_i, $1 \le i \le k+m+n$, é uma coleção XML cujo estado é dado por:

(i) Para $1 \le i \le k+m$, temos que:

$$\$C'_i = \{ \$t \mid \$t \text{ é um elemento } <e> \text{ de tipo T e } \exists \$c \in \$C_i,$$
$$\text{tal que } exp_i(\$c)=true \text{ e } \$t \equiv_{\mathcal{A}_{[T\&Ti]}} \$c\}.$$

(ii) Para $k+m+1 \le i \le k+m+n$, temos que:

$$\$C'_i = \{ \$t \mid \$t \text{ é um elemento } <e> \text{ de tipo T e } \exists \$c \in \$C_i$$
$$\text{tal que } \$t \equiv_{\mathcal{A}_{[T\&Ti]}} \$c \}.$$

É importante notar que as coleções $C'_1, ..., C'_k$ são compatíveis para as operações de união ($\cup$), as coleções $\{ C'_1 \cup ... \cup C'_k \}$, C'_{k+1}, ...e C'_{k+m} são compatíveis para as operações de *outer union* ($\overset{\circ}{\cup}_T$) e as coleções $\{ \{C'_1 \cup ... \cup C'_k \} \overset{\circ}{\cup}_T C'_{k+1} \overset{\circ}{\cup}_T ... \overset{\circ}{\cup}_T C'_{k+m} \} \}$, C'_{k+m+1}, ... e C'_{k+m+n} são compatíveis para as operações de *left outer union* ($\leftarrow_T$).

A assertiva global é definida com base nos relacionamento semântico da visão com as coleções bases. No caso da assertiva global ψ acima citada, para qualquer estado σ, são válidas as seguintes restrições.

(i) $\sigma(V) \subset \$C'_i$, $1 \le i \le k+m+n$.;

(ii) $\$C'_i \cap \$C'_j = \varnothing$, $1 \le i,j \le k$ e $j \ne i$, ($\$C'_i \cap \$C'_j$ são conjuntos disjuntos)

(iii) $C'_i ∩ {$C'_1 U ... U $C'_k} ≠ ∅, k+1≤i≤k+m (os conjuntos $C'_i e {$C_1 U ... U $C_k} não são disjuntos, ou seja, têm interseção);

(iv) $C'_i ⊂ {$C'_1 U ... U $C'_{k+m}}), k+m+1≤i≤k+m+n.

Note que a assertiva global de uma visão de integração envolve mais de uma coleção base e que sua forma visa integrá-las de uma maneira mais eficiente do que simplesmente realizar a operação de *outer union* entre todas as extensões das coleções bases. Observe que da restrição (iv), dado que $C'_i ⊂ {$C_1 U ... U $C_{k+m}}), k+m+1≤i≤k+m+n, $C'_i é um conjunto que não contribui com novos elementos para a visão, mas contribui com novas propriedades para os elementos dos outros conjuntos. Nesse caso a operação de *left outer union* pode ser usada, sendo esta mais eficiente do que a operação de *outer union*.

No exemplo a seguir, discutimos a definição de uma visão de integração de dados que integra autores das tabelas AUTORES_REL$_1$ e AUTORES_REL$_2$ dos esquemas relacionais AUTORES_BD$_1$ e AUTORES_BD$_1$ da Figura 3.24, respectivamente. Considere também o esquema relacional PERIODICOS_BD mostrado na Figura 3.24. Note que a relação PERIODICOS_REL é referenciada pela relação ANAIS_REL$_2$ através da ligação ℓ:*ANAIS_REL*$_2 \xrightarrow{f}$ *PERIODICOS_REL*, onde *f*:*ANAIS_REL*$_2${ISSN} → *PERIODICOS_REL*{ISSN}.

Considere a visão de integração AutoresIBM_V, cujo esquema é mostrado na Figura 3.23. A visão AutoresIBM_V integra dados de autores que trabalham na IBM. A assertiva global de AutoresIBM_V é dada por:

AutoresIBM_V ≡ *AUTORES_REL*$_2$[*instituição* = *'IBM'*] ⟖$_{TAutor\_V}$ *AUTORES_REL*$_1$,

Note que, apesar do esquema de relação *AUTORES_REL*$_1$ não possuir um atributo que determine a instituição de trabalho do autor, tal esquema contribui para visão com os dados dos livros dos autores. Ou seja, *AUTORES_REL*$_1$ não contribui com elementos para a visão, mas complementa os dados dos elementos da visão.

As assertivas de TAutor_V com *AUTORES_REL*$_1$ e *AUTORES_REL*$_2$ são mostradas nas Figuras 3.26 e 3.27, respectivamente. Note que a ligação ℓ permite que sejam especificadas as assertivas de TAnais_V com *PERIODICOS_REL*, como mostra a Figura 3.27. Com essas assertivas, é possível realizar a operação de *left outer join* entre os elementos <*autores*> obtidos de AUTORES_REL$_2$ com os elementos <*anais*> obtidos de PERIODICOS_REL.

Sejam $AUTORES_REL$_1$ e $AUTORES_REL$_2$ estados de *AUTORES_REL*$_1$ e *AUTORES_REL*$_2$ no estado σ, respectivamente (vide Figuras 3.4 e 3.5). O valor de AutoresIBM_V em σ, mostrado na Figura 3.28, é dado por:

$$\sigma(\text{AutoresIBM\_V}) = \text{AUTORES'}_2 \text{ Ł}_{\text{TAutor\_V}} \text{ AUTORES'}_1, \text{ onde}$$

$$\$\text{AUTORES'}_2 = \{ \$t \mid \$t \text{ é um elemento } \langle autor \rangle \text{ de tipo TAutor\_V e } \exists \$a_2 \in \$\text{AUTORES\_REL}_2,$$

$$\text{tal que } \$a_2/\text{instituicao} = \text{'IBM'} \text{ e } \$t \equiv_{\mathcal{A}[\text{TAutor\_V\&}TAUTORES\_REL1]} \$a_2 \}, \text{ e}$$

$$\$\text{AUTORES'}_1 = \{ \$t \mid \$t \text{ é um elemento } \langle autor \rangle \text{ de tipo TAutor\_V e } \exists \$a_1 \in \$\text{AUTORES\_REL}_1$$

$$\text{e } \exists \$a_1 \in \$\text{AUTORES'}_2, \text{ tal que } \$t[\text{email}] = \$a_2[\text{email}] \text{ e } \$t \equiv_{\mathcal{A}[\text{TAutor\_V\&}AUTORES\_REL2]} \$a_1 \}.$$

Os estados de AUTORES'$_2$ e AUTORES'$_1$ são mostrados nas Figuras 3.29 e 3.30, respectivamente. É importante notar que AUTORES'$_1$ e AUTORES'$_2$ são compatíveis para a operação *left outer union*.

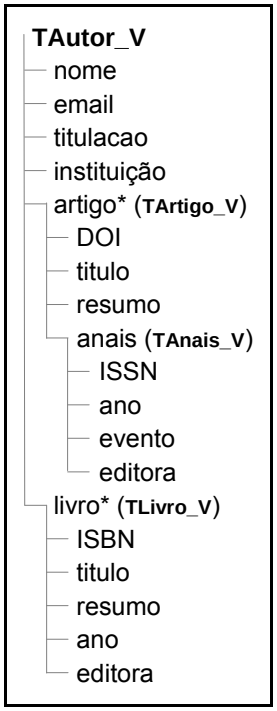

Figura 3.23. Tipo XML TAutor_V

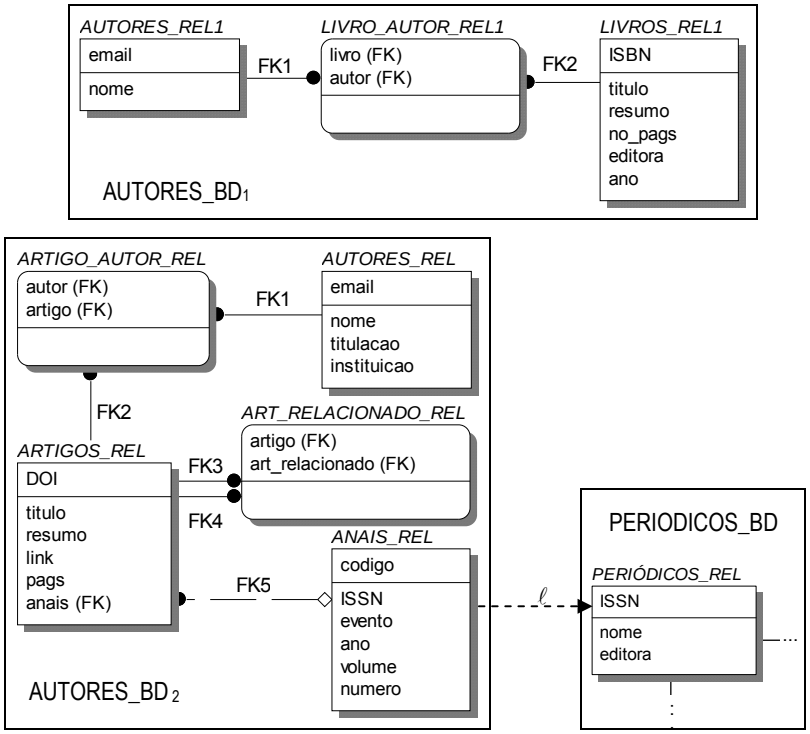

Figura 3.24. Esquema Relacionais AUTORES_BD$_1$, AUTORES_BD$_2$ e PERIODICOS_BD

PERIODICOS_REL

ISSN	NOME	EDITORA
0163-5808	ACM SIGMOD Record	ACM Press
1066-8888	The VLDB Journal	Springer-Verlag New York, Inc.
1-58113-507-6	Symposium on Principles of Database Systems	ACM Press
1049-331X	ACM Transactions on Software Engineering and Methodology	ACM Press

Figura 3.25. Estado de PERIODICOS_BD

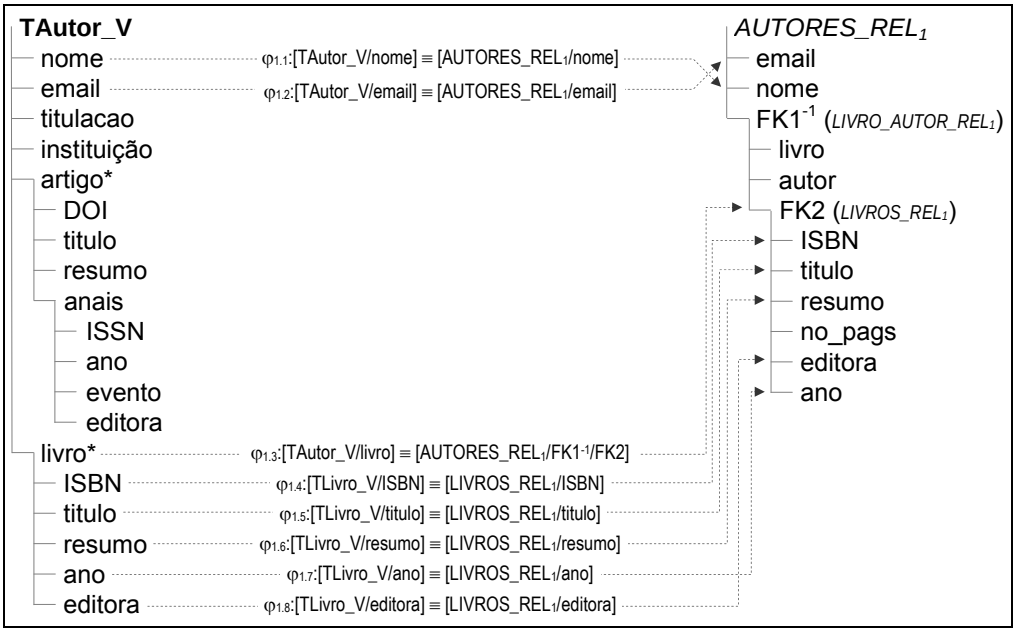

Figura 3.26. Assertivas de TAutor_V com *AUTORES_REL1*

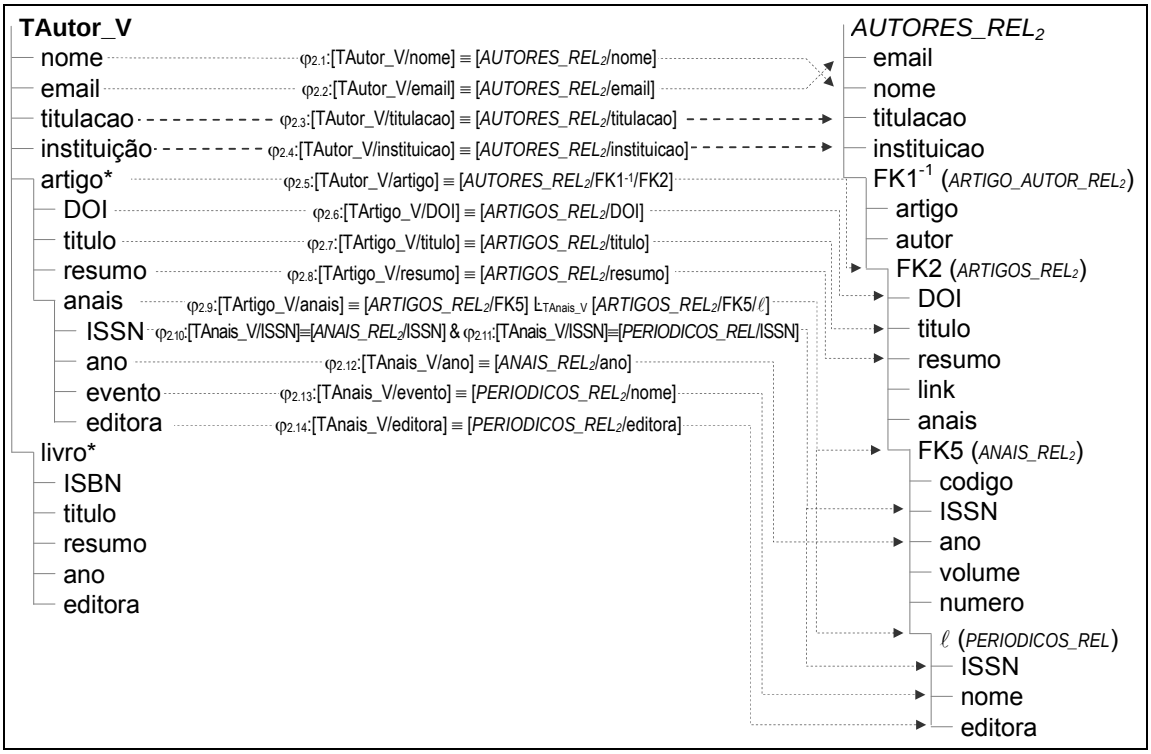

Figura 3.27. Assertivas de TAutor_V com *AUTORES_REL2*

```
<autor>
 <nome>Andrew Eisenberg</nome>
 <email>andrew.eisenberg@us.ibm.com</email>
 <titulacao>Ph.D.</titulacao>
 <instituicao>IBM</instituicao>
 <artigo>
   <DOI>10.1145/565117.565141</DOI>
   <titulo>SQL/XML is making good progress</titulo>
   <resumo>Not very long ago, we discussed...</resumo>
   <ano2003</ano>
 </artigo>
 <livro>
   <ISBN>1558605622</ISBN>
   <titulo>Understanding SQL and Java Together</titulo>
   <resumo>With the growth of Java and the rise...</resumo>
   <ano>2000</ano>
   <editora>Morgan Kaufmann</editora>
 </livro>
</autor>
```

```
<autor>
 <nome>Lucian Popa</nome>
 <email>lucian@almaden.ibm.com</email>
 <titulacao>Ph.D.</titulacao>
 <instituicao>IBM</instituicao>
 <artigo>
   <DOI>10.1O16/j.tcs.2004.10.033</DOI>
   <titulo>Data exchange: semantics and query answering</titulo>
   <resumo>Data exchange is the problem of...</resumo>
   <ano>2005</ano>
 </artigo>
</autor>
```

Figura 3.28. Estado da visão AutoresIBM_V

```
<autor>
 <nome>Andrew Eisenberg</nome>
 <email>andrew.eisenberg@us.ibm.com</email>
 <titulacao>Ph.D.</titulacao>
 <instituicao>IBM</instituicao>
 <artigo>
   <DOI>10.1145/565117.565141</DOI>
   <titulo>SQL/XML is making good progress</titulo>
   <resumo>Not very long ago, we discussed...</resumo>
   <ano2003</ano>
 </artigo>
</autor>
```

```
<autor>
 <nome>Lucian Popa</nome>
 <email>lucian@almaden.ibm.com</email>
 <titulacao>Ph.D.</titulacao>
 <instituicao>IBM</instituicao>
 <artigo>
   <DOI>10.1O16/j.tcs.2004.10.033</DOI>
   <titulo>Data exchange: semantics and query answering</titulo>
   <resumo>Data exchange is the problem of...</resumo>
   <ano>2005</ano>
 </artigo>
</autor>
```

Figura 3.29. Estado da coleção AUTORES'$_2$

```
<autor>
 <nome>Andrew Eisenberg</nome>
 <email>andrew.eisenberg@us.ibm.com</email>
 <livro>
   <ISBN>1558605622</ISBN>
   <titulo>Understanding SQL and Java Together</titulo>
   <resumo>With the growth of Java and the rise...</resumo>
   <ano>2000</ano>
   <editora>Morgan Kaufmann</editora>
 </livro>
</autor>
```

Figura 3.30. Estado da coleção AUTORES'$_1$

3.5 Geração do Plano de Materialização a partir das Assertivas de Correspondência da Visão

Nesta seção, apresentamos o algoritmo de geração do plano de materialização de uma visão de integração de dados a partir de suas assertivas de correspondência. No restante da seção, considere V uma visão de integração de dados, onde <e, T> é o esquema de V, $\mathcal{A}$ é o conjunto de assertivas de V em termos das fontes locais e K é uma chave para V. No que se segue, dada uma coleção XML C, $C denota o estado de C.

O algoritmo de geração do plano de materialização da visão V consiste de três passos:

Passo 1: Geração do *Workflow* Inicial de V: Neste passo, é gerado o *workflow* que computa o estado da visão com base na assertiva global da visão. O *workflow* gerado é descrito usando os *workflows* de cada fonte primária, os quais são gerados no Passo 2.

Passo 2: Geração dos *Workflows* dos Processos Primários: Neste passo, é gerado o *workflow* de cada fonte primária com base nas assertivas entre a visão e a fonte. O *workflow* gera elementos no formato da visão a partir dos dados da fonte primária.

Passo 3: Geração dos *Workflows* dos Processos Primários: Neste passo, é gerado o *workflow* de cada fonte secundária com base nas assertivas entre a visão e a fonte. O *workflow* gera, a partir dos dados da fonte secundária, elementos que irão complementar os elementos gerados por outros *workflows*.

Passo 1 – Geração do *Workflow* Inicial de V ($\mathcal{W}_i$[V])

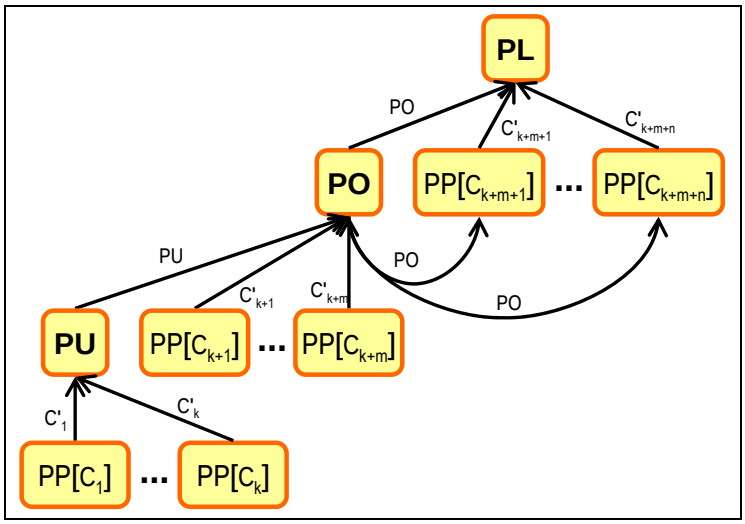

Figura 3.31. *Workflow* inicial $\mathcal{W}_i$[V]

O *workflow* inicial $\mathcal{W}_i$[V] é definido com base na assertiva global da visão. Seja

$$V \equiv ((C_1[exp_1] \cup ... \cup C_k[exp_k]) \dot{\cup} C_{k+1}[exp_{k+1}] \dot{\cup} ... \dot{\cup} C_{k+m}[exp_{k+m}]) \dot{\cup} C_{k+m+1} \dot{\cup} ... \dot{\cup} C_{k+m+n})$$

a assertiva global de V. O grafo do *workflow* $\mathcal{W}_i$[V] é mostrado na Figura 3.31, cujos processos são descritos a seguir:

- PP[C_i]: *Processo Primário* da coleção base C_i, $1 \leq i \leq k+m+n$.

  Para as coleções C_1, ..., C_{k+m}, o processo PP[C_i] retorna uma coleção XML C'_i, cujo estado é dado por:

$$\$C'_i = \{ \$t \mid \$t \text{ é um elemento } <e> \text{ de tipo T e } \exists \$c \in \$C_i,$$

$$\text{tal que } exp_i(\$c)=true \text{ e } \$t \equiv_{\mathcal{A}[T\&Ti]} \$c \}.$$

Para as coleções C_{k+m+1}, ..., C_{k+m+n}, o processo $PP[C_i]$ retorna uma coleção XML C'_i, cujo estado é dado por:

$$\$C'_i = \{ \$t \mid \$t \text{ é um elemento } <e> \text{ de tipo T e } \exists \$c \in \$C_i \text{ e } \exists \$p \in \$PO,$$
$$\text{tal que } \$p[K] = \$c[K] \text{ e } \$t \equiv_{\mathcal{A}[T\&Ti]} \$c \},$$

onde PO é a coleção XML resultante do processo PO, descrito mais adiante.

$PP[C_1]$, ..., $PP[C_{k+m+n}]$ são processos compostos, cujos *workflow*s são gerados no Passo 2.

- PU: *Processo União*. PU realiza a operação de união simples entre as coleções C'_i, resultantes dos processos $PP[C_i]$, $1 \leq i \leq k$. O resultado do processo PU é uma coleção XML PU, cujo estado é dado por:

  $$\$PU = (... ((\$C'_1 \cup \$C'_2) \cup \$C'_3) ... \cup \$C'_k).$$

  O processo PU pode ser implementado como um processo simples, que realiza uma seqüência de operações de união simples entre os conjuntos resultantes dos processos $PP[C_1]$, ..., $PP[C_k]$. Esse processo pode ser otimizado realizando algumas das operações de união em paralelo.

- PO: *Processo Outer Union*. PO realiza a operação de *outer union* entre as coleções PU, C'_{k+1}, ..., e C'_{k+m}. O resultado do processo PO é uma coleção XML PO, cujo estado é dado por:

  $$\$PO = (... ((\$PU \mathbin{\dot{\cup}_T} \$P'_{k+1}) \mathbin{\dot{\cup}_T} \$P'_{k+2}) ... \mathbin{\dot{\cup}_T} \$P'_{k+m}).$$

  Assim como o processo PU, processo PO pode ser implementado como um processo simples, que realiza a seqüência de operações descrita acima. Esse processo pode ser otimizado realizando algumas das operações de *outer union* em paralelo.

- PL: *Processo Outer Left Outer Union*. PL realiza a operação de *left outer union* entre as coleções PO, C'_{k+m+1}, ..., e C'_{k+m+n}. O resultado do processo PL é uma coleção XML PL, cujo estado é dado por:

  $$\$PL = (... ((\$PO \mathbin{⟕_T} \$C'_{k+m+1}) \mathbin{⟕_T} \$C'_{k+m+2}) ... \mathbin{⟕_T} \$C'_{k+m+n}).$$

  Assim como os processos PU e PO, processo PL pode ser implementado como um processo simples, que realiza a seqüência de operações descrita acima. Esse processo pode ser otimizado realizando algumas das operações de *left outer union* em paralelo.

De acordo com o *workflow* $\mathcal{W}_i[V]$ da Figura 3.31, temos que:

(i) Primeiro, os processos PP[C_1], ..., PP[C_{k+m}] são realizados em paralelo;

(ii) Em seguida, o processo PU integra os resultados dos processos PP[C_1], ..., PP[C_k];

(iii) Depois, o processo PO integra os resultados dos processos PU, PP[C_{k+1}], ..., PP[C_{k+m}];

(iv) Em seguida, os processos PP[C_{k+m+1}], ..., PP[C_{k+m+n}] são realizados em paralelo; e

(v) Finalmente, o processo PL integra os resultados dos processos PU, PP[C_{k+m+1}], ..., PP[C_{k+m+n}].

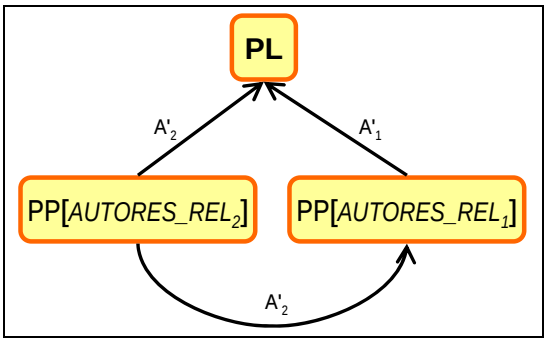

Figura 3.32. *Workflow* $\mathcal{W}_i$[V] de Autores_V

O *workflow* inicial $\mathcal{W}_i$[Autores_V] da visão Autores_V da Seção 4 é representado pelo grafo da Figura 3.32. Neste grafo, temos que:

- O processo PP[*AUTORES_REL2*] retorna uma coleção XML A'_2, cujo estado é dado por:

  $A'_2 = \{ \$a'_2 \mid \$a'_2$ é um elemento <*autor*> de tipo TAutor_V e $\exists a_2 \in \$AUTORES\_REL_2$, tal que a_2/instituicao=*instituicao*$_P$ e $\$a'_2 \equiv_{\mathcal{A}[TAutor\_V\&AUTORES\_REL2]} a_2 \}$.

- O processo PP[*AUTORES_REL1*] retorna uma coleção XML A'_1, cujo estado é dado por:

  $A'_1 = \{ \$a'_1 \mid \$a'_1$ é um elemento <*autor*> de tipo TAutor_V e $\exists a_1 \in \$AUTORES\_REL_1$ e $\exists \$a'_2 \in A'_2$, tal que $\$a'_1$[email] = $\$a'_2$[email] e $\$a'_1 \equiv_{\mathcal{A}[TAutor\_V\&AUTORES\_REL1]} a_1 \}$.

  A geração dos *workflows* dos processos PP[*AUTORES_REL2*] e PP[*AUTORES_REL1*] é mostrada no Passo 2.

- O processo PL realiza a operação *left outer union* entre as coleções resultantes dos processos PP[*AUTORES_REL2*] e PP[*AUTORES_REL1*]. O processo PL retorna uma coleção XML PL, cujo estado é dado por:

  $$\$PL = \$A'_2 \text{ L}_{TAutor} \$A'_1.$$

De acordo com o *workflow* $\mathcal{W}_i$[Autores_V]: O processo PP[*AUTORES_REL2*] é realizado e, em seguida, o processo PP[*AUTORES_REL1*] é realizado. Por fim, o processo PL integra os resultados.

Passo 2 – Geração dos *Workflows* dos Processos Primários PP[C_i], $1 \leq i \leq k+m+n$

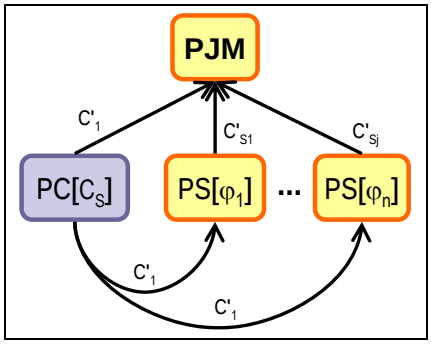

Figura 3.33. *Workflow* do Processo Primário PP[C_i]

Seja exp_i a condição de seleção de C_i na assertiva global de V, caso exista. Sejam $\varphi_1, ..., \varphi_p$ as assertivas de junção de C_i, dadas por:

$$\varphi_j:[T'/e'] \equiv [T'_j/\delta_j] \text{ L}_{Te'} [T'_j/\delta_j/\ell_j],\ 1 \leq j \leq p,$$

onde

- $\ell_j:C_i(\lambda_j) \xrightarrow{f} C_{Sj}$ e $f:T_{\delta j}[a_1, ..., a_k] \rightarrow T_{Sj}[b_1,..., b_k]$,
- C_{Sj} é uma coleção secundária de V,
- λ_j é um caminho composto do caminho de T_i correspondente ao contexto de T'_j (caminho que vai do elemento primário de C_i até o elemento de tipo T'_j que faz correspondência com o elemento de tipo T) mais o caminho δ_j (vide Figura 3.34).

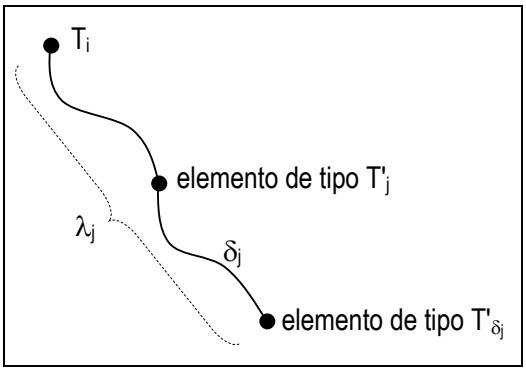

Figura 3.34. Caminho λ_j

O *workflow* do Processo Primário de C_i é representado pelo grafo da Figura 3.33, cujos processos são descritos a seguir:

- PC[C_i]: *Processo Consulta*. PC[C_i] realiza uma consulta Q sobre o esquema da coleção C_i, cujo resultado é uma coleção XML C'_i. O estado de C'_i é dado por:

  (i) Para $1 \leq i \leq k+m$, temos que:

$$\$C'_i = \{ \$t \mid \$t \text{ é um elemento <e> de tipo T e } \exists \$c \in \$C_i,$$

$$\text{tal que } \$t \equiv_{\mathcal{A}^*_{[T\&Tp]}} \$c \text{ e } exp_P(\$c)=true \}.$$

(ii) Para k+m+1≤i≤k+m+n, temos que:

$$\$C'_i = \{\ \$t \mid \$t \text{ é um elemento <e> de tipo T e } \exists \$c \in \$C_i \text{ e } \exists \$p \in \$PO,$$
$$\text{tal que } \$t \equiv_{\mathcal{A}^*_{[T\&Tp]}} \$c \text{ e } \$t[K] = \$p[K]\ \},$$

onde K é chave comum de PO e C_i.

A consulta Q é gerada automaticamente pelos algoritmos da Seção 3.6, com base nas ACs entre V e C_i. No caso de C_i ser um tabela relacional, o algoritmo GeraConsultaSQL/XML (Figura 3.40) gera uma consulta SQL/XML. No caso de C_i ser uma coleção XML, o algoritmo GeraConsultaXQuery (Figura 3.42) gera uma consulta XQuery.

- PS[φ_j]: *Processo Secundário* de φ_j, 1≤j≤p. PS[φ_j] recebe como parâmetro a coleção C'_i e retorna uma coleção XML C'_{Sj}, cujo estado é dado por:

$$\$C'_{Sj} = \{\ \$t \mid \$t \text{ é um elemento <e'> de tipo } T_{e'} \text{ e } \exists \$c_S \in \$C_{Sj} \text{ e } \exists \$c \in \$C'_i/\lambda_j,$$
$$\text{tal que } \$c_S \in \$c/\delta_j/\ell_j \text{ e } \$t \equiv_{\mathcal{A}_{[Te\&TSj]}} \$c_S\ \},$$

onde $T_{e'}$ é o tipo de e' em φ_j.

Os *workflows* dos processos PS[φ_1], ..., PS[φ_p] são gerados no Passo 3.

- PJM: *Processo Junção Múltipla*. PJM realiza a operação de *left outer join* entre a coleção C'_i e as coleções C'_{S1}, ..., e C'_{Sp}. O resultado do processo PJM é uma coleção XML PJM, cujo estado é dado por:

$$\$PJM = (\ ...\ (\ (\ \$C'_i\ \hat{J}_{(T_{e'},\delta_1)}\ \$C'_{S_1}\ )\ \hat{J}_{(T_{e'},\delta_2)}\ \$C'_{S_2}\ )\ ...\ \hat{J}_{(T_{e'},\delta_n)}\ \$C'_{S_p}\ ).$$

Caso não exista nenhuma assertiva de junção φ_j, o Processo Primário de C_i será um processo simples que consiste apenas da consulta Q sobre C_i.

De acordo com o *workflow* da Figura 3.33, temos que:

(i) O processo PC[C_i] realiza a consulta Q sobre C_i;

(ii) Em seguida, os processos PS[ψ_j], 1≤j≤p, são realizados.

(iii) Por fim, o processo PMJ integra os resultados do processo PC[C_i] com os resultados dos processos PS[ψ_j], 1≤j≤p.

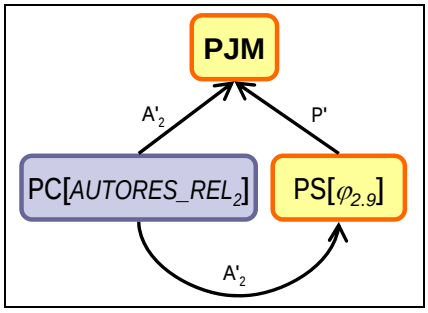

Figura 3.35. *Workflow* do processo PP[AUTORES_REL$_2$]

O *workflow* do processo PP[*AUTORES_REL*$_2$] da visão Autores_V é representado pelo grafo da Figura 3.35. Neste grafo, temos que:

- O processo PC[*AUTORES_REL*$_2$] realiza a consulta SQL/XML da Figura 3.36, a qual retorna uma coleção XML A'$_2$, cujo estado é dado por:

  $A'_2 = { $a'_2 | $a'_2 é um elemento <*autor*> de tipo TAutor_V e $\exists a_2 \in$ $AUTORES\_REL_2$,

  tal que $a'_2 \equiv_{\mathcal{A}^*_{[TAutor\_V\&AUTORES\_REL2]}}$ a }.

- O processo PS[φ$_{2.9}$] é o processo secundário para a assertiva de junção

  $\varphi_{2.9}$:[TArtigo_V/anais] ≡ [*ARTIGOS_REL*$_2$/FK5] $\lfloor_{TAnais\_V}$ [*ARTIGOS_REL*$_2$/FK5/ℓ],

  onde ℓ:*AUTORES_REL*$_2$(*FK1*$^{-1}$/*FK2*/*FK5*) $\xrightarrow{f}$ *PERIODICOS_REL*.

  PS[φ$_{2.9}$] retorna uma coleção XML $P', cujo estado é dado por:

  $P' = { $p' | $p' é um elemento <*anais*> de tipo TAnais_V e $\exists p \in$ $PERIODICOS\_REL$

  e $\exists a \in$ A'_2/artigo/anais, tal que p/ISSN = a/ISSN e $p' $\equiv_{\mathcal{A}_{[TAnais\_V\&PERIODICOS\_REL]}}$ p }.

- O processo PJM retorna uma coleção XML J, cujo estado é dado por:

  $J = $A'_2 $\hat{J}_{(TAnais,artigos/anais)}$ $P'.

O processo PP[*AUTORES_REL*$_1$] da visão Autores_V é um processo simples, pois não existe assertiva de junção para a relação AUTORES_REL$_1$. Deste modo, o processo PP[*AUTORES_REL*$_1$] consiste unicamente da consulta mostrada na Figura 3.37 sobre AUTORES_REL$_1$.

```
SELECT XMLElement("autor",
  XMLForest("nome", AU.nome),
  XMLForest("email", AU.email),
  XMLForest("titulacao", AU.titulacao),
  XMLForest("instituicao", AU.instituicao),
  (SELECT XMLAgg( XMLElement("artigo",
     XMLForest("DOI", AR.DOI),
     XMLForest("titulo", AR.titulo),
     XMLForest("resumo", AR.resumo),
     (SELECT XMLElement("Anais",
        XMLForest("ISSN", AN.ISSN),
        XMLForest("ano", AN.ano) )
        FROM ANAIS_REL AN
        WHERE AR.anais = AN.codigo) ) )
   FROM ARTIGO_AUTOR_REL AA, ARTIGOS_REL AR
   WHERE AA.autor = AU.email AND
         AA.artigo = AR.DOI) )
FROM AUTORES_REL AU
WHERE AU.instituicao = instituicaoP
```

Figura 3.36. Consulta SQL/XML sobre AUTORES_REL2

```
SELECT XMLElement("autor",
  XMLForest("nome", AU.nome),
  XMLForest("email", AU.email),
  (SELECT XMLAgg( XMLElement("livro",
     XMLForest("ISBN", LI.ISBN),
     XMLForest("titulo", LI.titulo),
     XMLForest("resumo", LI.resumo),
     XMLForest("ano", LI.ano),
     XMLForest("editora", LI.editora)
   ) )
   FROM LIVRO_AUTOR_REL1 LA,
        LIVROS_REL1 LI
   WHERE LA.livro = AU.email AND
         LA.livro = LI.ISBN) )
FROM AUTORES_REL1 AU
WHERE AU.email IN $A'/email
```

Figura 3.37. Consulta SQL/XML sobre AUTORES_REL1

Passo 3 – Geração dos *Workflows* dos Processos Secundários PS[φ]

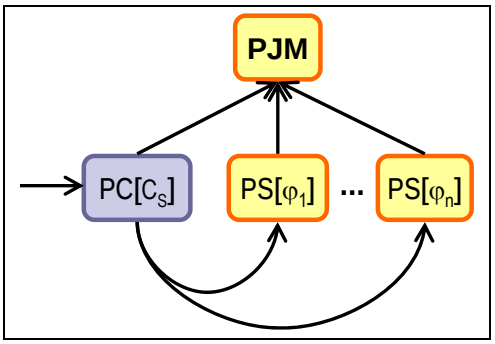

Figura 3.38. *Workflow* do processo secundário PS[φ]

Seja φ:[T'/e'] ≡ [T'$_C$/δ] Ł$_{Te'}$ [T'$_C$/δ/ℓ] a assertiva de junção do Processo Secundário em questão, onde:

- ℓ:C(λ) $\xrightarrow[f]{}$ C_S e *f*:T_δ[a_1, ..., a_k] → T_S[b_1,..., b_k];
- λ é um caminho composto do caminho de T_C correspondente ao contexto de T'$_C$ mais o caminho δ.
- C é uma coleção primária ou secundária de V;
- C_S é uma coleção primária ou secundária de V.

Sejam φ_1, ..., φ_n as assertivas de junção de C_S, dadas por:

$$\varphi_i:[T''/e''] \equiv [T'_S/\delta_i] \text{ Ł}_{Te''} [T'_S/\delta_i/\ell_i],\ 1 \leq i \leq n,$$

onde

- ℓ_i:C_S(δ_i) $\xrightarrow[f]{}$ C_{Si} e *f*:$T_{\delta i}$[a_1, ..., a_k] → T'$_{Si}$[b_1,..., b_k];
- λ_j é um caminho composto do caminho de T_S correspondente ao contexto de T'$_S$ mais o caminho δ_j;
- C_{Si} é um esquema de coleção secundária de V.

O *workflow* do Processo Secundário PS[φ] é representado pelo grafo da Figura 3.38, cujos processos são descritos a seguir:

- PC[C_S]: *Processo Consulta*. PC[C_S] recebe uma coleção E como parâmetro, realiza uma consulta *Q* sobre a coleção C_S e retorna uma coleção XML C'$_S$, cujo estado é dado por:

  $C'_S = \{ \$s' \mid \$s'$ é um elemento <e'> de tipo $T_{e'}$ e ∃$s∈$C_S$ e ∃$e∈$E(δ),

  tal que $s' ≡$_{A*[Tc&Ts]}$ $s e $s'[K'] = $e[K'] },

  onde $T_{e'}$ é o tipo de e' em φ e K' é chave comum de C_S e E(δ).

A consulta Q é gerada automaticamente pelos algoritmos da Seção 2.6. com base nas ACs entre V e C_S. No caso de C_S ser uma tabela relacional, o algoritmo GeraConsultaSQL/XML (Figura 3.40) gera uma consulta SQL/XML. No caso de C_S ser uma coleção XML, o algoritmo GeraConsultaXQuery (Figura 3.42) gera uma consulta XQuery.

- PS[φ_i]: *Processo Secundário* da assertiva φ_i. PS[φ_i] recebe como parâmetro uma coleção C'_S e retorna uma coleção XML C'_{Si}, cujo estado é dado por:

$$\$C'_{Si} = \{ \$t \mid \$t \text{ é um elemento <e"> de tipo } T_{e"} \text{ e } \exists \$c_S \in \$C_{Si} \text{ e } \exists \$c_S \in \$C'_S,$$
$$\text{tal que } \$c_S/\delta_i/\ell_i = \{\$c_S\} \text{ e } \$t \equiv_{\mathcal{A}_{[Te\&Tsi]}} \$c_S,\ 1 \le i \le n \},$$

  onde $T_{e"}$ é o tipo de e" em φ_i.

  Os *workflows* dos processos PS[φ_1], ..., PS[φ_n] são gerados de forma recursiva pelo Passo 3.

- PJM: *Processo Junção Múltipla*. PJM realiza a operação de *left outer join* entre a coleção C'_S e as coleções C'_{S1}, ..., e C'_{Sn}. O resultado do processo PJM é uma coleção XML PJM, cujo estado é dado por:

$$\$PJM = (... ((\$C'_S \hat{J}_{(T_{e"},\delta_1)} \$C'_{S_1}) \hat{J}_{(T_{e"},\delta_2)} \$C'_{S_2}) ... \hat{J}_{(T_{e"},\delta_n)} \$C'_{S_n}).$$

Caso não exista nenhuma assertiva de junção φ_i, o Processo Secundário será um processo simples que consiste apenas da consulta Q sobre C_S.

De acordo com o *workflow* da Figura 3.38:

(i) O processo PC[C_S] realiza a consulta Q sobre C_S;

(ii) Em seguida, os processos PS[φ_i], $1 \le i \le n$, são realizados.

(iii) Por fim, o processo PMJ integra os resultados do processo PC[C_S] com os resultados dos processos PS[φ_i], $1 \le i \le n$.

O processo PS[$\varphi_{2.9}$] da visão Autores_V é um processo simples, pois não existe assertiva de junção para a relação PERIODICOS_REL. Deste modo, o processo PS[$\varphi_{2.9}$] consiste unicamente da consulta mostrada na Figura 3.39 sobre PERIODICOS_REL.

```
SELECT XMLElement("Anais",
   XMLForest("ISSN", PE.ISSN),
   XMLForest("evento", PE.nome),
   XMLForest("editora", PE.editora) )
FROM PERIODICOS_REL PE
WHERE PE.ISSN IN $A'"/artigo/anais
```

Figura 3.39. Consulta SQL/XML sobre PERIODICOS_REL

3.6 Algoritmos GeraConsultaSQL/XML e GeraConsultaXQuery

Nesta seção, apresentamos os algoritmos GeraConsultaSQL/XML e GeraConsultaXQuery, os quais são usados na geração das consultas dos processos PC.

3.6.1 Algoritmo GeraConsultaSQL/XML

O algoritmo GeraConsultaSQL/XML, mostrado na Figura 3.40 gera, a partir das assertivas de correspondência da VN, uma consulta SQL/XML que materializa o conteúdo da VN a partir dos dados do banco. O algoritmo é semelhante ao algoritmo de geração da definição SQL/XML de visões XML apresentado na Seção 2.7. A diferença neste caso é que temos que tratar as expressões de seleção de visões parametrizadas. No algoritmo GeraConsultaSQL/XML, o algoritmo TraduzFiltro (Figura 3.41) traduz a expressão de seleção da VN em uma expressão SQL equivalente. O algoritmo TraduzFiltro trata expressões de seleção com parâmetros, as quais serão utilizadas pelas visões do Capítulo 4.

ENTRADA: tipo XML T; rótulo e; esquema de relação *R*; conjunto $\mathcal{A}$ de assertivas que especificam completamente T em termos de *R*; expressão de seleção $\pi = (exp_1 \wedge ... \wedge exp_n)$, onde $n \geq 1$
SAÍDA: consulta SQL Q

```
Seja Q uma string;

Q := "SELECT XMLElement(\"e\", " + GeraConstrutorSQL/XML(T, R, 𝒜, r) + " )
       FROM R r ";
Se π ≠ ∅ Então
    Q := Q + "WHERE ";
    Para cada expi de π Faça
        Q := Q +  TraduzFiltro(expi, 𝒜, r) + " AND ";
    Fim Para
Fim Se

Retorne Q;
```

Figura 3.40. Algoritmo GeraConsultaSQL/XML

ENTRADA: expressão $\pi = (e_1/.../e_n\ \theta\ v)$, onde $n \geq 1$, θ é um operador booleano e v é um valor atômico ou um parâmetro; conjunto $\mathcal{A}$ de assertivas de correspondência; alias r para o esquema de relação R
SAÍDA: expressão SQL Π

Seja $[T/e_1] \equiv [R/\delta_1]$ a assertiva de e_1 em $\mathcal{A}$;
  $[Te_i/e_{i+1}] \equiv [Re_i/\delta_{i+1}]$ a assertiva de e_i em $\mathcal{A}$, $1 \leq i \leq n-2$;
  $[Te_{n-1}/e_n] \equiv [Re_{n-1}/\delta_n.exp]$ a assertiva de e_n em $\mathcal{A}$, tal que exp = a ou $exp = \{a_1,...,a_k\}$;
Seja $\delta := \delta_1. \ldots .\delta_n.$exp;
Seja Π uma string;

No caso
  Caso 1: Se δ = a Então
    Π := "(r.a ϕ 'v')", onde ϕ é o operador SQL equivalente a θ;
  Caso 2: Se $\delta = \{a_1,...,a_k\}$ Então
    Π := "((r.a_1 ϕ 'v') OR ... OR (r.a_k ϕ 'v'))", onde ϕ é o operador SQL equivalente a θ;
  Caso 3: Se $\delta = \varphi_1. \ldots .\varphi_m$.a, onde φ_1 é um link de R para R_1 e φ_i é um link de R_{i-1} para R_i, $1 \leq i \leq m$, Então
    Π := "EXISTS (SELECT * FROM Joinφ(r) AND (r_n.a ϕ 'v'))" ,
    onde ϕ é o operador SQL equivalente a θ;
  Caso 4: Se $\delta = \varphi_1. \ldots .\varphi_m.\{a_1,...,a_k\}$ Então
    Π := "EXISTS (SELECT * FROM Joinφ(r) AND ((r_n.a_1 ϕ 'v') OR ... OR (r_n.a_k ϕ 'v')))" ,
    onde ϕ é o operador SQL equivalente a θ;
Fim caso
Retorne Π;

Nota:
Joinφ(r) é o fragmento SQL definido na Seção 2.7.

Figura 3.41. Algoritmo TraduzFiltro

Por exemplo, considere o processo PP[$AUTORES\_REL_2$] mostrado na Figura 3.35, o qual realiza a consulta Q da Figura 3.36 sobre $AUTORES\_REL_1$. Considere o conjunto de assertivas $\mathcal{A}_{[TAutor\_V\&AUTORES\_REL2]}$ mostrado na Figura 3.27, o qual especifica completamente TAutor_V em termos de $AUTORES\_REL_2$. A consulta SQL/XML Q é gerada por

GeraConsultaSQL/XML(TAutor_V, $AUTORES\_REL_2$, $\mathcal{A}_{[TAutor\_V\&AUTORES\_REL2]}$, "instituição = *IBM*").

De acordo com o algoritmo GeraConsultaSQL/XML, inicialmente é gerada a função construtora $\tau[AUTORES\_REL_2 \rightarrow TAutor\_V][r]$, a qual constrói o conteúdo estendido dos elementos da visão. Em seguida, é chamado o algoritmo TraduzFiltro para traduzir a expressão "instituição = *IBM*" (Caso 1 do algoritmo).

3.6.2 Algoritmo GeraConsultaXQuery

O algoritmo GeraConsultaXQuery mostrado na Figura 3.42 gera, a partir das assertivas de correspondência da VN, uma consulta XQuery que materializa o conteúdo da VN. O algoritmo é baseado no algoritmo de geração da definição SQL/XML de visões XML

apresentado na Seção 2.7. A diferença neste caso é que a coleção base é XML e, portanto, o construtor do conteúdo estendido dos elementos da VN é composto de subconsultas XQuery sobre a coleção XML base.

ENTRADA: tipo XML T, rótulo e, esquema de coleção XML C=<e_C, T_C>, conjunto $\mathcal{A}$ de assertivas que especificam completamente T em termos de T_C, expressão de seleção π = ($exp_1 \wedge ... \wedge exp_n$), onde $n \geq 1$
SAÍDA: consulta XQuery Q

```
Seja Q uma string;

Q := "for $c in doc(\"C\")/eC";
Se π ≠ ∅ Então
    Q := Q + " where ";
    Para cada expi = (δ θ v) de π, onde δ é um caminho de T, θ é um operador booleano e v é um valor
    atômico ou um parâmetro, Faça
        Q := Q +  " $c/δ θ' \"v\" ", onde θ' é o operador XQuery equivalente a θ;
    Fim Para
Fim Se
Q := Q + "return <e " + GeraConstrutorXQuery(T, TC, A, $c) + "</e>";

Retorne Q;
```

Figura 3.42. Algoritmo GeraConsultaXQuery

Input: um tipo XML T; um tipo XML T_C; um conjunto de assertivas de correspondência $\mathcal{A}$ que especifica completamente T em termos de T_C, uma variável $c para C=<e,$T_C$>
Output: Função $\tau[C \rightarrow T][\$c]$

```
Seja τ uma string;
τ := ∅;

Para cada atributo a de tipo T onde ψa é a AC para a em A Faça
    τ := τ + GeraSubconsultaSQL/XML(A, φa, r);
Fim Para;

τ := τ + ">"

Para Cada elemento e de T onde ψe é a AC para e em A Faça
      τ := τ + GeraSubconsultaSQL/XML(A, ψe, $c);
Fim Para;

Retorne τ ;
```

Figura 3.43. Algoritmo GeraConstrutorXQuery

ENTRADA: um conjunto $\mathcal{A}$ de assertivas de correspondência que especificam completamente T em termos de T_C, uma AC $[T/e] \equiv [T_C/\delta]$ em $\mathcal{A}$ onde e é um elemento ou atributo de tipo T_e, e uma variável $c para C=<e,$T_C$>
SAÍDA: uma subconsulta XQuery

Seja Q uma string;
No caso
  Caso 1: Se e é um elemento de ocorrência simples, T_e é um tipo simples e δ = a, Então
    Q := "<e>{$c/a/text()}</e>";
  Caso 2: Se e é um elemento de ocorrência simples, T_e é um tipo simples e $\delta = \varphi$/a, Então
    Q := "<e>{$c/$\varphi$/a/text()}</e>";
  Caso 3: Se e é um elemento de ocorrência múltipla, T_e é um tipo simples e $\delta = \{a_1,...,a_n\}$, Então
    Q := "<e>{$c/$a_1$/text()}</e>" + ... + "<e>{$c/$a_n$/text()}</e>";
  Caso 4: Se e é um elemento de ocorrência múltipla, T_e é um tipo simples $\delta = \varphi/\{a_1,...,a_n\}$, Então
    Q := "<e>{$c/$\varphi$/$a_1$/text()}</e>" + ... + "<e>{$c/$\varphi$/$a_n$/text()}</e>";
  Caso 5: Se e é um elemento de ocorrência múltipla, T_e é um tipo simples $\delta = \varphi$/ a, Então
    Q := "for $c' in $c/$\varphi$/a return <e>{$c'/a/text()}</e>";
  Caso 6: Se e é um elemento de ocorrência simples, T_e é um tipo complexo e $\delta = \varphi$, Então
    Q := "<e>{ " + GeraConstrutorXQuery(T_e, T_δ, $\mathcal{A}$, $c/$\varphi$) + " }</e>";
  Caso 7: Se e é um elemento de ocorrência múltipla, T_e é um tipo complexo $\delta = \varphi$, Então
    Q := "for $c' in $c/$\varphi$ return <e>{ " + GeraConstrutorXQuery(T_e, T_δ, $\mathcal{A}$, $c/$\varphi$) + " }</e>";
  Caso 8: Se e é um elemento de ocorrência simples, T_e é um tipo complexo δ = NULL, Então
    Q := "<e>{ " + GeraConstrutorXQuery(T_e, T_δ, $\mathcal{A}$, $c) + " }</e>";
  Caso 9: Se e é um atributo e δ = a, Então
    Q := " e=\"$c/a\" ";
  Caso 10: Se e é um atributo e $\delta = \varphi$/a, Então
    Q := " e=\"$c/$\varphi$/a\" ";
Fim Caso;
Retorne Q ;

Figura 3.44. Algoritmo GeraSubconsultaXQuery

3.7 Conclusões

Neste capítulo, mostramos que podemos especificar completamente uma visão XML de integração de dados em termos de mais de uma coleção base usando assertivas de correspondência. Como visto, as assertivas definem um mapeamento de instâncias das coleções base em instâncias do esquema da visão XML.

Este capítulo apresenta duas principais contribuições: (i) Extensão do formalismo proposto no Capítulo 2 para especificar visões XML sobre múltiplas fontes de dados; (ii) Desenvolvimento de algoritmos que geram, baseados nas ACs da VN, o plano de materialização que computa o conteúdo da visão.

CAPÍTULO 4
Especificação e Geração de Visões de Navegação para Aplicações *Web*

Neste capítulo, apresentamos um processo para especificação e geração de visões de navegação para aplicações Web de intenso acesso e manipulação de dados (aplicações DIWA). As visões de navegação (VNs) são implementadas como uma visão XML cujo conteúdo é extraído de uma ou múltiplas fontes de dados. Estas visões são especificadas usando o formalismo proposto no Capítulo 3. A vantagem do uso de assertivas para especificação das VNs é que a geração e a manutenção das VNs podem ser realizadas de forma automática.

O capítulo está organizado da seguinte forma: A Seção 4.1 discute sobre aplicações Web e visões de navegação; A Seção 4.2 discute visões de navegação parametrizadas; A Seção 4.3 apresenta um processo para especificação e geração de visões de navegação para aplicações Web sobre uma única base; A Seção 4.4 apresenta um processo para especificação e geração de visões de navegação para aplicações Web sobre múltiplas bases; A Seção 4.5 apresenta o algoritmo GeraVLEs, utilizado nos processos das Seções 4.3 e 4.4; A Seção 4.6 apresenta as conclusões.

4.1 Introdução

Sistemas de informação vêm utilizando o paradigma da *Web* (e tecnologias) para prover aos seus usuários informações contidas em diferentes bases. Geralmente, essas aplicações possuem um grande número de páginas, cujos conteúdos são dinamicamente extraídos de uma ou mais fontes de dados, e que requerem intenso acesso e atualização dos dados (aplicações DIWA) [7]. Neste trabalho, os requisitos de conteúdo de uma página dinâmica de uma aplicação *Web* são especificados através de uma visão XML, a qual denominamos Visão de Navegação (VN).

Como discutido em [7], a especificação e geração das VNs não é uma tarefa fácil pois, em geral, os conteúdos das VNs são gerados por módulos compilados, os quais

representam uma grande quantidade de código *ad-hoc* específico da aplicação e da plataforma. Essa complexidade se traduz em um alto custo de desenvolvimento e manutenção. Além disso, no caso de mudanças nos requisitos da aplicação ou nos esquemas das fontes de dados, uma grande quantidade de VNs pode ser afetada, o que torna necessária a reconstrução dos códigos de implementação dessas VNs. A especificação, construção e manutenção das VNs se tornam ainda mais complexas quando seus conteúdos são extraídos de múltiplas fontes de dados.

As metodologias de especificação e geração de aplicações *Web* mais recentes utilizam uma arquitetura de três níveis de esquemas como mostrado na Figura 4.1 Nessa arquitetura, as visões de navegação são especificadas sobre o Esquema Conceitual da Aplicação (ECA), o qual representa uma visão integrada dos requisitos de conteúdo da aplicação. O ECA, por sua vez, é especificado sobre os esquemas das fontes locais. Essa arquitetura define uma camada de independência lógica entre os requisitos de cada página e os dados das fontes locais, tornando mais fácil a especificação, geração e manutenção das VNs.

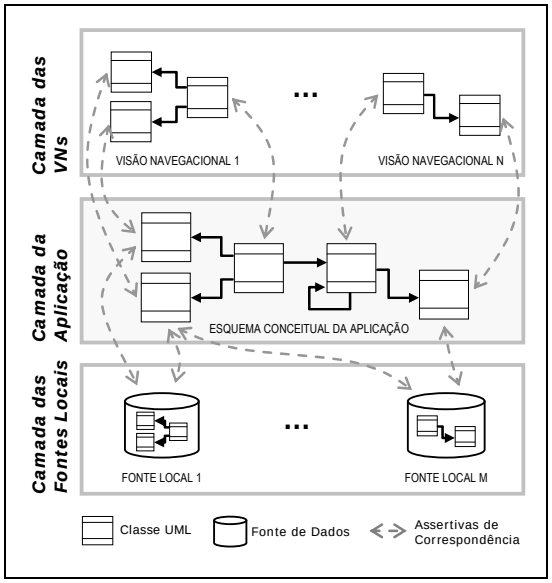

Figura 4.1. Arquitetura de Esquemas da Aplicação

Neste capítulo, apresentamos um processo para especificação e geração de VNs para aplicações *Web* cujo conteúdo é extraído de uma ou mais fontes de dados. Em nossa abordagem, uma VN é especificada com o auxílio de um conjunto de assertivas de correspondência entre o esquema da VN e os esquemas das fontes de dados. Como visto nos Capítulos 2 e 3, o formalismo das assertivas é adequado para esse tipo de aplicação, uma vez que as assertivas especificam de forma axiomática como os

elementos da visão de navegação são sintetizados a partir dos elementos das fontes locais. Além disso, o formalismo é independente de linguagem de consulta e permite que as definições das VNs sejam geradas automaticamente.

4.2 Visões de Navegação Parametrizadas

Na realização de uma tarefa em uma aplicação DIWA, muitas vezes a página a ser exibida tem conteúdo gerado de acordo com a ação realizada pelo usuário na página anterior. Por exemplo, o usuário entra com um valor em um formulário ou seleciona um link e a página seguinte tem conteúdo gerado de acordo com essa ação. Deste modo, a dinamicidade dessas páginas não é determinada somente pelo estado do banco de onde os dados são extraídos, como também por um ou mais parâmetros cujos valores dependem da navegação do usuário pela aplicação. Neste caso, a visão de navegação da página é implementada como uma visão XML parametrizada, cujos parâmetros são definidos com base na troca de dados existente entre as páginas.

Uma visão XML parametrizada é uma quádrupla $V = \langle S, \mathcal{P}, \psi, \mathcal{A} \rangle$, onde:

(i) $S = \langle e, T \rangle$ é o esquema da visão;

(ii) $\mathcal{P} = \{@param_1, ..., @param_j\}$ é uma lista de parâmetros;

(iii) ψ é uma assertiva de correspondência global da forma:

$$V \equiv ((C_1[exp_1] \cup ... \cup C_k[exp_k]) \mathbin{\mathring{\cup}_T} C_{k+1}[exp_{k+1}] \mathbin{\mathring{\cup}_T} ... \mathbin{\mathring{\cup}_T} C_{k+m}[exp_{k+m}]) \mathbin{\dot{-}_T} C_{k+m+1} \mathbin{\dot{-}_T} ... \mathbin{\dot{-}_T} C_{k+m+n})$$

onde C_i é uma coleção base, a qual pode ser uma tabela relacional, tabela de objetos ou coleção XML, e exp_i é uma expressão predicativa [10] da forma:

$$exp_i = (\delta_1 \theta_1 \text{ '}@param_1\text{'} \wedge ... \wedge \delta_j \theta_j \text{ } @param_j\text{'}),$$

onde $\delta_1, ..., \delta_j$ são caminhos de C_i e $\theta_1, ..., \theta_j$ são operadores booleanos.

(iv) $\mathcal{A} = \bigcup_{i=1}^{k+m+n} \mathcal{A}_i$, onde $\mathcal{A}_i$ é um conjunto de assertivas de correspondência que especifica completamente T em termos de T_i (o tipo de C_i), $1 \leq i \leq k+m+n$.

A seguir, é apresentado um exemplo de visão de navegação parametrizada. Considere a aplicação *AutoresWeb*, a qual disponibiliza informações sobre autores. Suponha que *AutoresWeb* extraia informações da fonte de dados AUTORES_BD, cujo esquema é mostrado na Figura 4.2

Dentre as atividades que podem ser realizadas com a aplicação, considere a tarefa *"Consultar dados de um autor"*, ilustrada na Figura 4.3 O UID que representa

essa tarefa é mostrado na Figura 4.4 Nesta tarefa, a página inicial (4.3(a)) mostra uma lista de nomes de autores. O usuário pode então selecionar um autor, e o sistema retorna as informações sobre o autor selecionado. Assim, essa tarefa possui duas VNs, $\mathcal{V}_1$ e $\mathcal{V}_2$, cujos esquemas são mostrados nas Figuras 4.5 e 4.6, respectivamente. Note que, $\mathcal{V}_2$ é uma visão parametrizada, cujo parâmetro representa o código do autor selecionado em $\mathcal{V}_1$.

Suponha que conteúdo de $\mathcal{V}_2$ seja extraído da tabela AUTORES_REL da fonte AUTORES_BD. Assim, temos que:

- $\{cod_P\}$ é lista de parâmetros de $\mathcal{V}_2$;
- $\mathcal{V}_2 \equiv$ ((*AUTORES_REL*[codigo = @cod_P]) é assertiva global de $\mathcal{V}_2$;
- As assertivas de $\mathcal{V}_2$ com *AUTORES_REL* são mostradas na Figura 4.6.

As consultas Q_1 e Q_2 mostradas na Figura 4.7 extraem o conteúdo das VNs $\mathcal{V}_1$ e $\mathcal{V}_2$, respectivamente. Em Q_2, *@cod* é o parâmetro da consulta. Como discutido na Seção 3.6, tais consultas são geradas automaticamente pelo algoritmo GeraConsultaSQL/XML. Como veremos, o formalismo das assertivas automatiza grande parte do processo de especificação e geração das VNs.

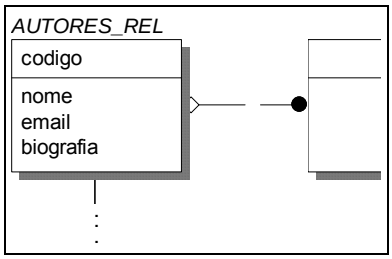

Figura 4.2. Esquema relacional da fonte AUTORES_BD

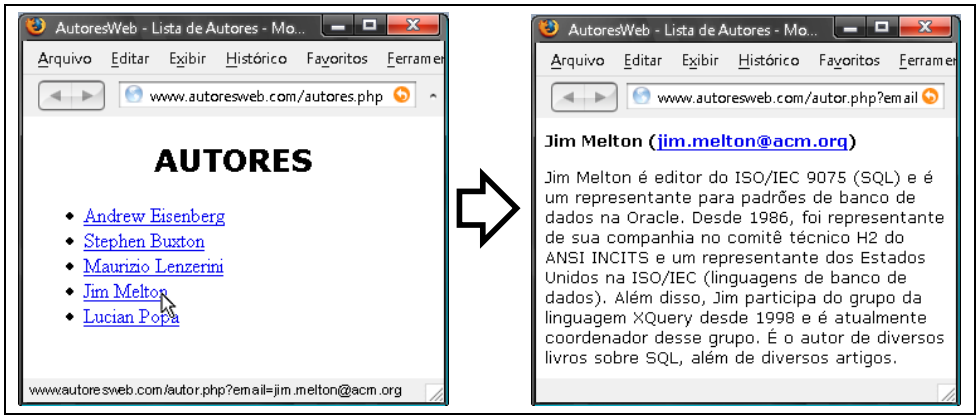

Figura 4.3. Tarefa *"Consultar dados de um autor"*

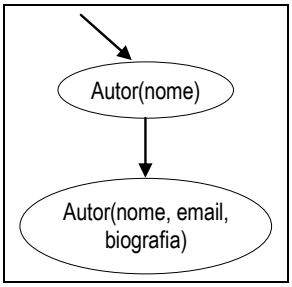

Figura 4.4. UID da tarefa *"Consultar dados de um autor"*

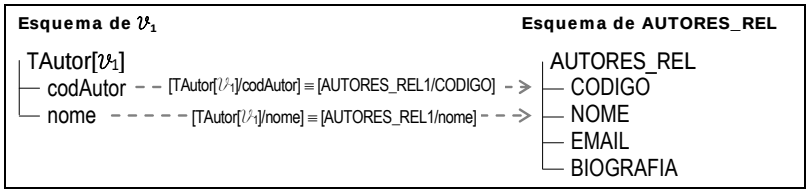

Figura 4.5. Assertivas de Correspondência de $\mathcal{V}_1$ com a fonte AUTORES_BD

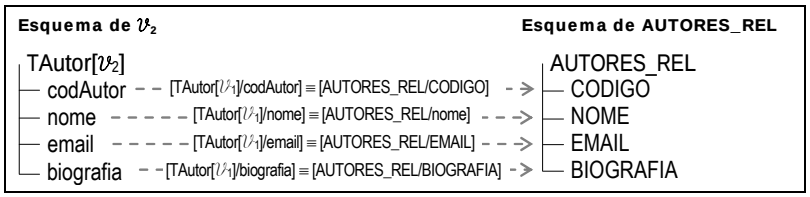

Figura 4.6. Assertivas de Correspondência de $\mathcal{V}_2$ com a fonte AUTORES_BD

Q1:
```
SELECT XMLELEMENT("autor",
   XMLFOREST(A.CODIGO AS "codAutor"),
   XMLFOREST(A.NOME AS "nome")
)
FROM AUTORES_REL A;
```

Q2:
```
SELECT XMLELEMENT("autor",
   XMLFOREST(A.CODIGO AS "codAutor"),
   XMLFOREST(A.NOME AS "nome"),
   XMLFOREST(A.EMAIL AS "email"),
   XMLFOREST(A.BIOGRAFIA AS "biografia"))
FROM AUTORES_REL A
WHERE A.CODIGO = '@cod';
```

Figura 4.7. Consultas Q_1 e Q_2 sobre a fonte AUTORES_BD

Na restante desse capítulo, apresentamos dois processos para a especificação e geração das VNs de uma aplicação *Web*. O primeiro processo é mais simples, pois trata aplicações cujo conteúdo é extraído de uma única fonte de dados. Neste caso, as VNs são definidas por uma única consulta SQL/XML ou XQuery, dependendo da natureza das fontes. O segundo processo trata aplicações que extraem dados de mais de uma fonte de dados. Neste caso, as VNs são definidas pelo plano de materialização da visão.

4.3 Visões de Navegação Definidas sobre uma Única Base

Nesta seção, é proposto um processo para sistematizar as tarefas de especificação, implementação e manutenção das VNs para aplicações DIWA que extraem seus dados de uma única base.

O processo proposto consiste de duas fases: (i) Projeto Conceitual e (ii) Projeto Navegacional. No *Projeto Conceitual* são obtidos os esquemas conceituais das VNs e o Esquema Conceitual da Aplicação (vide Figura 4.1). No *Projeto Navegacional* são geradas as consultas que computam os conteúdos das VNs.

Este processo parte dos cenários e casos de uso da aplicação para obter as VNs. Os cenários e casos de uso podem ser obtidos com base nas informações dos usuários e em documentação existente.

No restante dessa seção, considere a aplicação *WebBib*, a qual disponibiliza informações sobre autores e suas publicações. Suponha que a aplicação *WebBib* acesse informações da fonte AUTORES_BD, cujo esquema é mostrado na Figura 4.8. Para essa aplicação, considere a tarefa "*Consultar publicações de um autor*", cujo caso de uso é mostrado na Figura 4.9.

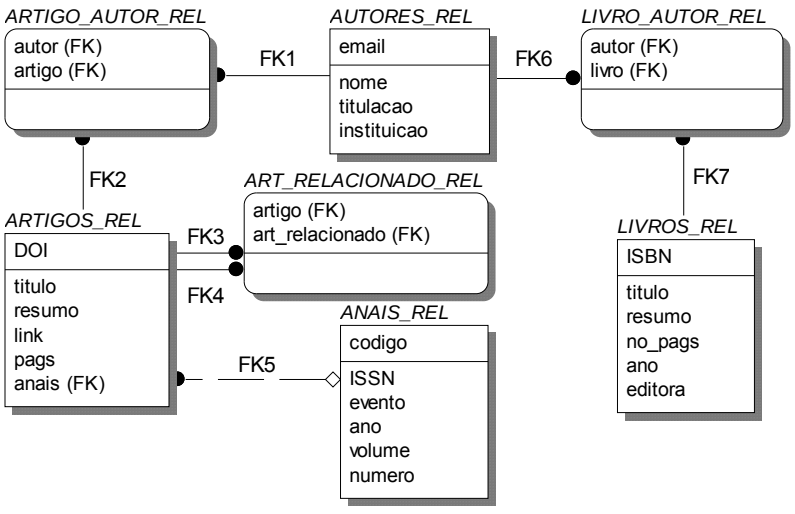

Figura 4.8. Esquema da fonte AUTORES_BD

Caso de Uso: *Consultar publicações de um autor*
Descrição:
1. O usuário informa o nome ou parte do nome do autor;
2. A aplicação retorna uma lista dos nomes dos autores que possuem o nome informado;
3. O usuário seleciona um autor;
4. A aplicação retorna o nome, titulação, uma lista de artigos e uma lista de livros do autor selecionado;
5. O usuário seleciona um artigo ou livro do autor;
6. Caso tenha selecionado um artigo, a aplicação retorna o título, resumo, dados dos autores, dados dos anais, páginas do artigo nos anais, lista de artigos relacionados e um link para *download* do artigo. Se desejar, o usuário pode obter informações de um artigo relacionado (neste caso, retorna para o item 6);
7. Caso tenha selecionado um livro, a aplicação retorna o título, o resumo, os dados dos autores, o ano de publicação, a editora, e o número de páginas.

Figura 4.9. Caso de uso *"Consultar publicações de um autor"*

A seguir descrevemos os detalhes das fases do processo proposto.

4.3.1 Projeto Conceitual

O Projeto Conceitual é realizado em três passos:

1. Modelagem dos Esquemas das Visões de Navegação
2. Geração do Esquema Conceitual da Aplicação
3. Geração do Mapeamento entre o ECA e o esquema da fonte de dados

Passo 1: Modelagem dos esquemas das Visões de Navegação

Para cada caso de uso especificado para a aplicação, é gerado um Diagrama de Interação dos Usuários (UID). UIDs são utilizados para representar as interações entre usuários e a aplicação durante a realização de uma tarefa descrita por um caso de uso. Os UIDs foram propostos para suprir as deficiências encontradas no emprego de casos de uso, que, por serem descrições textuais, costumam causar ambigüidades pela falta de precisão e concisão. UID é uma ferramenta que descreve apenas as trocas de informações entre os usuários e a aplicação, sem considerar aspectos relacionados com interface do usuário. Em [49] é apresentado como mapear um caso de uso no seu UID correspondente.

Considere o UID $\mathcal{U}$ mostrado na Figura 4.10, gerado a partir do caso de uso da Figura 4.9. Cada elipse de $\mathcal{U}$ representa uma interação entre o usuário e o sistema. A interação inicial de $\mathcal{U}$, representada por uma seta sem origem, indica que o sistema espera que o usuário informe o nome ou parte do nome do autor que deseja consultar. Em seguida, o sistema apresenta um conjunto de elementos Autor que possuem o nome

informado, e nome é apresentado como item de dados (...Autor(nome)). O usuário pode, então, selecionar um dos autores apresentados e o sistema retorna dados do autor selecionado, e assim por diante. O resultado de cada interação que causa processamento do sistema deve ser representado por uma elipse separada, conectada à interação origem por uma seta.

Cada interação do UID possui associada uma VN, a qual especifica os requisitos de conteúdo no contexto dessa interação. Por exemplo, os dados apresentados pelo sistema durante a segunda interação de $\mathcal{U}$ – ou seja, o nome dos autores – estão associados a uma visão sobre o banco de dados, a qual denominamos $\mathcal{V}_1$. Assim, o UID $\mathcal{U}$ possui quatro VNs: $\mathcal{V}_1$, $\mathcal{V}_2$, $\mathcal{V}_3$ e $\mathcal{V}_4$.

Neste momento, devemos especificar alguns dados que irão compor a definição das VNs: nome do elemento primário, tipo da VN, chaves e parâmetros. O nome do elemento primário da VN fica a cargo do projetista. O tipo da VN é derivado da estrutura descrita na interação correspondente, como apresentado em [26]. As chaves e os parâmetros são definidos com base no UID, como apresentado em [26].

Os esquemas das VNs $\mathcal{V}_1$, $\mathcal{V}_2$, $\mathcal{V}_3$ e $\mathcal{V}_4$ são mostrados nas Figuras 4.11, 4.12, 4.13 e 4.14, respectivamente. Tabela 4.1 mostra o restante dos dados das VNs.

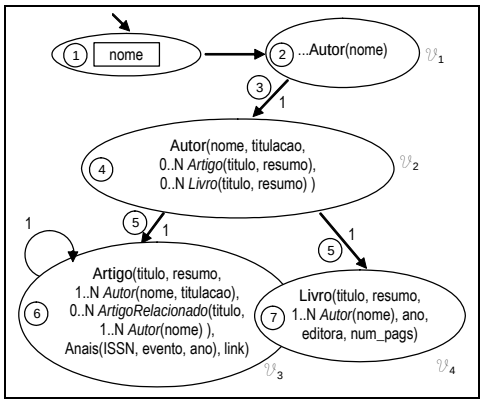

Figura 4.10. UID $\mathcal{U}_1$

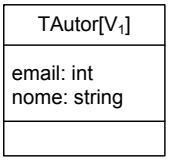

Figura 4.11. Esquema de $\mathcal{V}_1$

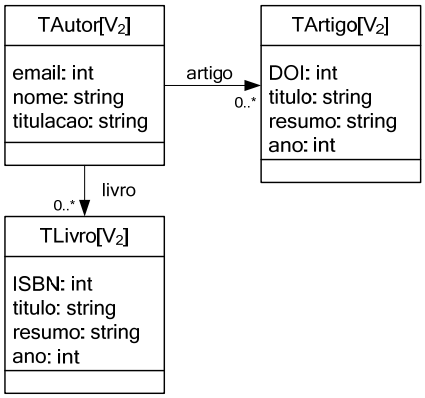

Figura 4.12. Esquema de $\mathcal{V}_2$

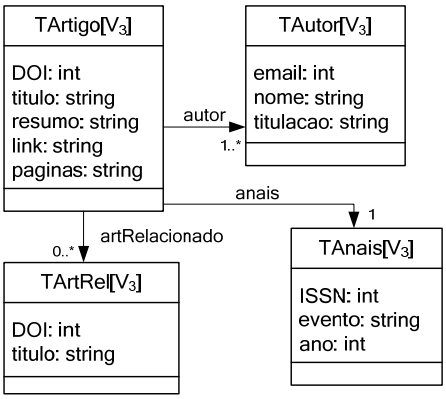

Figura 4.13. Esquema de $\mathcal{V}_3$

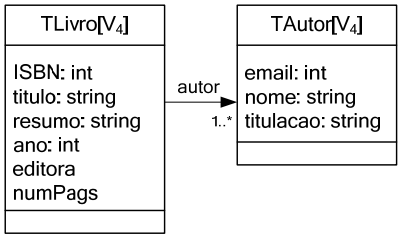

Figura 4.14. Esquema de $\mathcal{V}_4$

Tabela 4.1. Dados das VNs $\mathcal{V}_1$, $\mathcal{V}_2$, $\mathcal{V}_3$ e $\mathcal{V}_4$.

VN	Elemento Primário	Chave	Parâmetros
$\mathcal{V}_1$	autor	Email	{@*nome*}
$\mathcal{V}_2$	autor	Email	{@*email*}
$\mathcal{V}_3$	artigo	DOI	{@*DOI*}
$\mathcal{V}_4$	livro	ISBN	{@*ISBN*}

Passo 2: Geração do Esquema Conceitual da Aplicação

O ECA é obtido através da integração dos esquemas das VNs, de forma que este satisfaça os requisitos de conteúdo de todas as VNs, como proposto em [26]. De acordo com [26], as ACs das VNs com o ECA são geradas de forma interativa durante a integração das VNs. O tipo de uma VN está associado a um tipo do ECA, o qual denominamos *tipo base da* VN. Note que um tipo do ECA pode ser tipo base de várias VNs.

Integrando os esquemas das visões $\mathcal{V}_1$, $\mathcal{V}_2$, $\mathcal{V}_3$ e $\mathcal{V}_4$ obtemos o ECA mostrado na Figura 4.15. O tipo TAutor é tipo base das VNs $\mathcal{V}_1$ e $\mathcal{V}_2$. Os tipos TArtigo e TLivro são tipos base das VNs $\mathcal{V}_3$ e $\mathcal{V}_4$, respectivamente. No restante dessa seção, focaremos o exemplo na VN $\mathcal{V}_2$. As ACs de $\mathcal{V}_2$ com o ECA são mostrados na Figura 4.16.

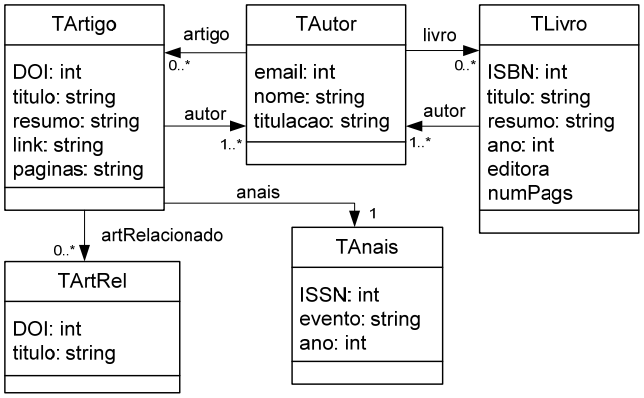

Figura 4.15. ECA de *WebBib*

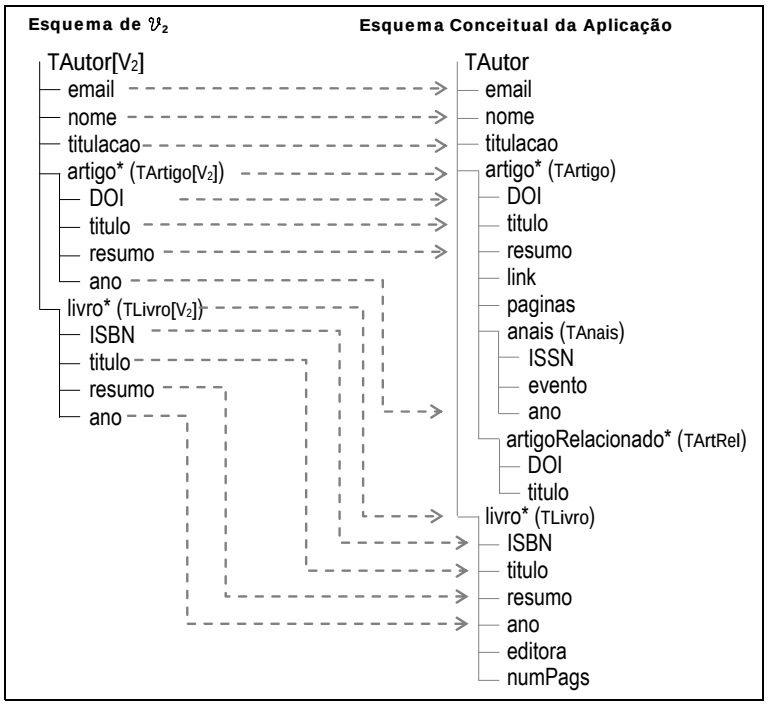

Figura 4.16. Assertivas de Correspondência de $\mathcal{V}_2$ com o ECA

Passo 3: Geração do Mapeamento entre o ECA e o esquema da fonte de dados

As assertivas do ECA são obtidas através do *matching* do ECA com o esquema da fonte de dados usando o processo proposto no Capítulo 2. As ACs do ECA com o esquema da fonte de dados são mostradas na Figura 4.17.

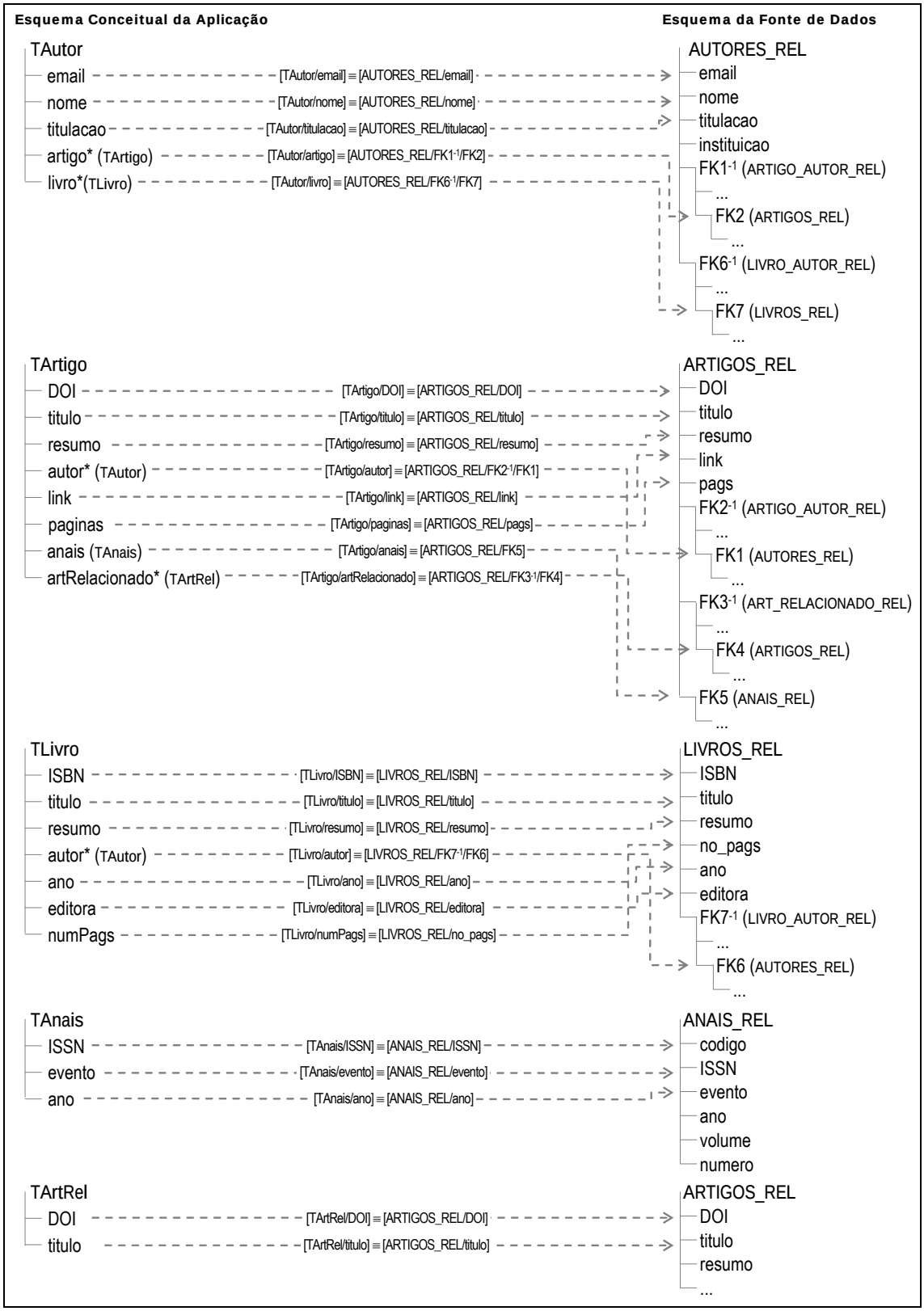

Figura 4.17. Assertivas de Correspondência do ECA com a fonte AUTORES_BD

4.3.2 Projeto Lógico

A fase do Projeto Lógico consiste da geração das consultas das VNs a partir das assertivas de correspondência. Consideramos duas soluções para a geração da consulta de uma VN:

(i) A consulta é definida sobre o esquema da fonte de dados.

(ii) A consulta é definida sobre o esquema de uma visão XML exportada pela fonte de dados, a qual tem estrutura compatível com a estrutura da VN. Essa solução é apropriada quando o banco possui mecanismo de visão XML, como discutido no Capítulo 2. Como veremos, a consulta XQuery que define a visão é bem simples, uma vez que o esquema da visão exportada é compatível com o esquema da VN. Outra vantagem é que mudanças no esquema da fonte não afetam as VNs, tornando a aplicação de fácil manutenção.

A seguir discutimos as duas soluções para a geração das consultas das VNs:

Solução (i): *Consulta é definida sobre o esquema da fonte de dados*

Neste caso, as ACs da VN especificam o mapeamento entre o esquema da VN e os esquema da fonte de dados, como mostra . Essas assertivas são derivadas das ACs da VN com o ECA e das ACs do ECA com o esquema da fonte de dados, as quais foram definidas no Projeto Conceitual.

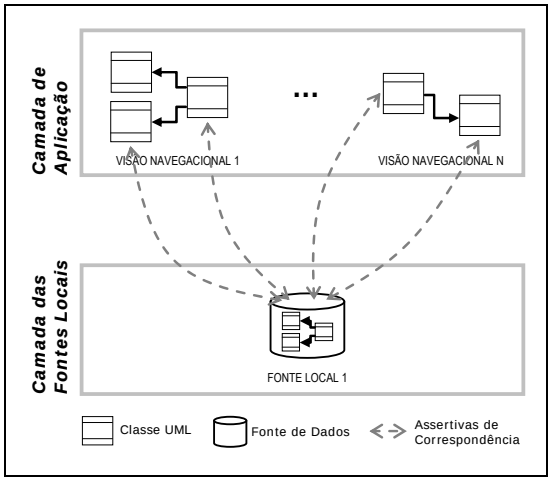

Figura 4.18. Arquitetura de esquemas da solução (i)

A consulta da VN é gerada automaticamente a partir das ACs da VN, como descrito no Capítulo 2. No caso de fonte relacional ou objeto-relacional, a consulta é definida em SQL/XML. No caso de fonte XML, a consulta é definida em *XQuery*.

Figura 4.19 mostra as assertivas da VN $\mathcal{V}_2$ com a fonte AUTORES_BD. A consulta Q gerada automaticamente a partir das assertivas é mostrada na Figura 4.20.

Nesta seção, a implementação das VNs é exemplificada através do uso do *framework* XSQL Pages. O Oracle XSQL Pages Publishing Framework [31][29] dá suporte à geração dinâmica de dados XML a partir de dados objeto-relacionais. Uma página XSQL é um documento XML composto de consultas SQL parametrizadas ou não que extraem informações de bancos de dados relacionais ou objeto-relacionais. A página é processada a cada requisição *Web* e seu resultado é apresentado no formato XML.

A implementação da VN $\mathcal{V}_2$ como uma página XSQL é apresentada na Figura 4.21. A expressão {@email} na cláusula WHERE denota um parâmetro de nome email, que corresponde ao email do autor sobre o qual se deseja obter os detalhes. A cada requisição a essa página, a expressão {@email} será substituída pelo valor do parâmetro email passado na requisição. A consulta presente na página é gerada com base nas assertivas da VN com a fonte de dados.

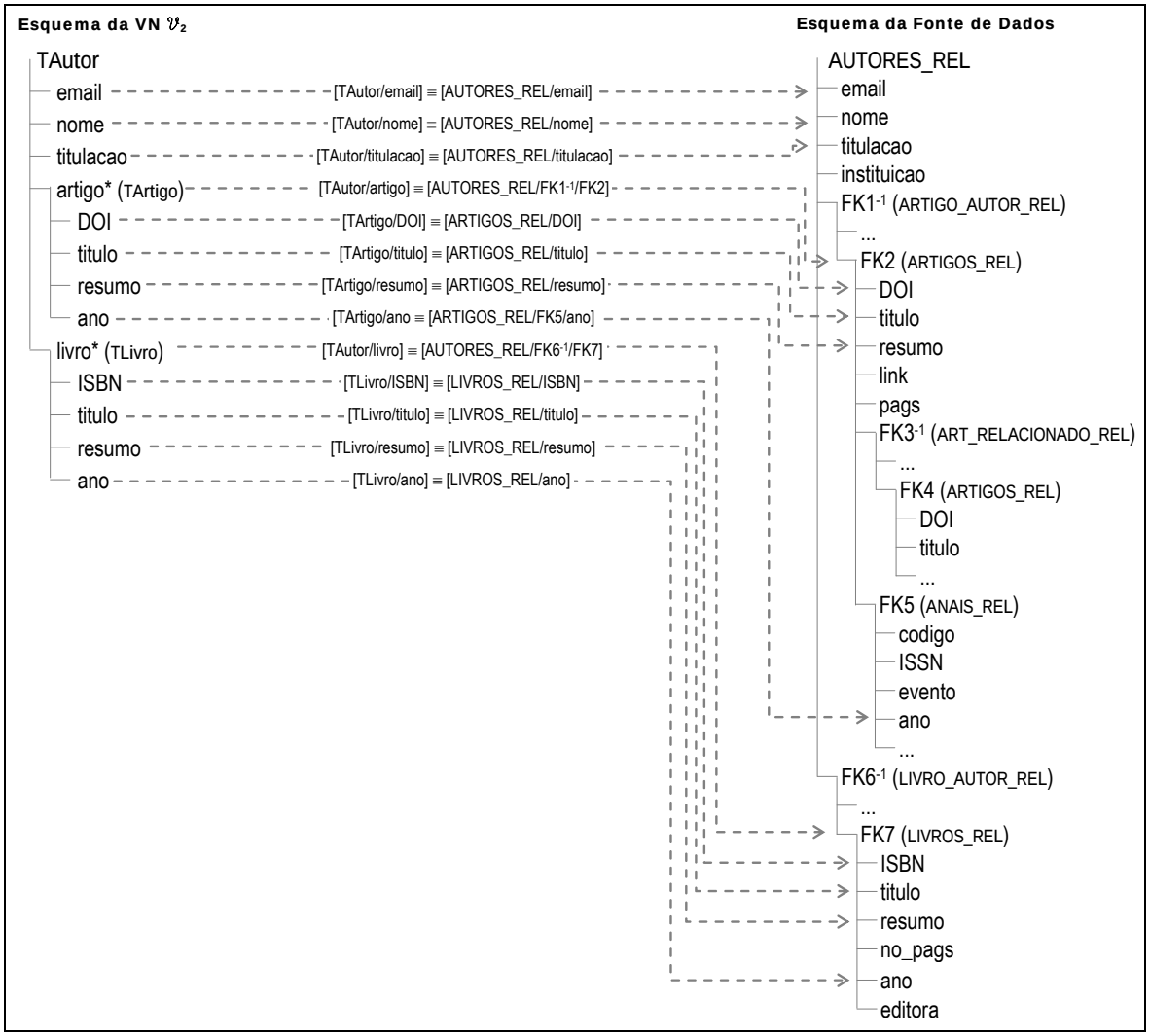

Figura 4.19. Assertivas de $\mathcal{V}_2$ com a fonte AUTORES_BD

```
SELECT XMLElement("autor",
  XMLForest("nome", AU.nome),
  XMLForest("email", AU.email),
  XMLForest("titulacao", AU.titulacao),
  (SELECT XMLAgg( XMLElement("artigo",
     XMLForest("DOI", AR.DOI),
     XMLForest("titulo", AR.titulo),
     XMLForest("resumo", AR.resumo),
     XMLForest("ano", (SELECT AN.ano FROM ANAIS_REL AN
                       WHERE AR.anais = AN.codigo) ) ) )
   FROM ARTIGO_AUTOR_REL AA, ARTIGOS_REL AR
   WHERE AA.autor = AU.email AND
         AA.artigo = AR.DOI),
  (SELECT XMLAgg( XMLElement("livro",
     XMLForest("ISBN", LI.ISBN),
     XMLForest("titulo", LI.titulo),
     XMLForest("resumo", LI.resumo),
     XMLForest("ano", LI.ano ) ) )
   FROM LIVRO_AUTOR_REL LA, LIVROS_REL LI
   WHERE LA.autor = AU.email AND
         LA.livro = LI.ISBN) )
FROM AUTORES_REL AU WHERE AU.email = '@email'
```

Figura 4.20. Consulta SQL/XML Q gerada a partir das assertivas de $\mathcal{V}_2$

```
<xsql:include-xml connection="conn"
                  xmlns:xsql="urn:oracle-xsql">

   SELECT XMLElement("autor",
     XMLForest("nome", AU.nome),
     XMLForest("email", AU.email),
     XMLForest("titulacao", AU.titulacao),
     (SELECT XMLAgg( XMLElement("artigo",
        XMLForest("DOI", AR.DOI),
        XMLForest("titulo", AR.titulo),
        XMLForest("resumo", AR.resumo),
        XMLForest("ano", (SELECT AN.ano FROM ANAIS_REL AN
                          WHERE AR.anais = AN.codigo) ) ) )
      FROM ARTIGO_AUTOR_REL AA, ARTIGOS_REL AR
      WHERE AA.autor = AU.email AND
            AA.artigo = AR.DOI),
     (SELECT XMLAgg( XMLElement("livro",
        XMLForest("ISBN", LI.ISBN),
        XMLForest("titulo", LI.titulo),
        XMLForest("resumo", LI.resumo),
        XMLForest("ano", LI.ano ) ) )
      FROM LIVRO_AUTOR_REL LA, LIVROS_REL LI
      WHERE LA.autor = AU.email AND
            LA.livro = LI.ISBN) ).getClobVal()
   FROM AUTORES_REL AU
   WHERE AU.email = '{@email}'

</xsql:include-xml>
```

Figura 4.21. Página XSQL que implementa a VN $\mathcal{V}_2$

Solução (ii): *Consulta é definida sobre o esquema da visão local exportada*

Neste caso, para cada tipo do ECA que seja tipo base de alguma VN, é gerada uma visão XML sobre o esquema do banco, denominada *Visão Local Exportada* (VLE). Tais visões minimizam o problema de heterogeneidade estrutural entre as VNs e o banco, caso exista. Além disso, as VLEs definem uma camada lógica tal que alterações no esquema do banco não afetam as VNs. Denominamos *visão base da* VN a visão local exportada cujo tipo é o tipo base da VN. Note que uma VLE pode ser visão base de mais de uma VN.

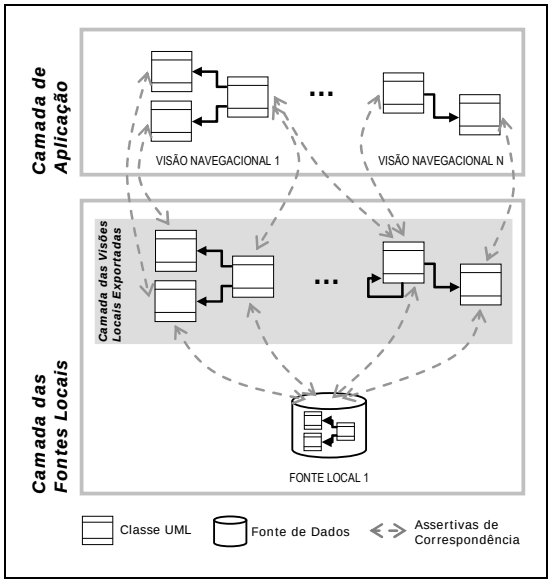

Figura 4.22. Arquitetura de Esquemas da Solução (ii)

Mais formalmente, seja T um tipo do ECA que é tipo base da VN $\mathcal{V}$=<$S_{\mathcal{V}}$, $\mathcal{P}_{\mathcal{V}}$, $\psi_{\mathcal{V}}$, $\mathcal{A}_{\mathcal{V}}$ >, onde $S_{\mathcal{V}}$=<$e_{\mathcal{V}}$, $T_{\mathcal{V}}$> e $\mathcal{A}_{\mathcal{V}}$ é o conjunto de assertivas entre a $T_{\mathcal{V}}$ e T. Seja $\mathcal{A}_{[T\&Tc]}$ o conjunto de assertivas de T com a coleção base C, onde T_C é o tipo de C. A visão local exportada V_E gerada a partir de $\mathcal{V}$ é dada por V_E=<S, ∅, ψ, $\mathcal{A}_{[T\&Tc]}$>, onde S=<$e_{\mathcal{V}}$, T> e ψ:[V ≡ C]. No caso de fonte relacional ou objeto-relacional que possui mecanismo de criação de visões SQL/XML, a visão exportada é gerada automaticamente a partir das ACs $\mathcal{A}_{[T\&Tc]}$ e ψ, como descrito no Capítulo 2.

As assertivas da VN com o esquema de visão base são obtidas diretamente das ACs da VN com o ECA (), uma vez que o tipo da visão exportada é o tipo base da VN. A consulta da VN é gerada automaticamente a partir das ACs da VN, usando o algoritmo da Seção 3.6.

Figura 4.23 mostra as assertivas da visão local exportada Autores_Exp = <S, {}, [TAutor ≡ *AUTORES_REL*], $\mathcal{A}_{[TAutor\&AUTORES\_REL]}$>, onde S=<autor, TAutor>, a qual é visão base das VNs $\mathcal{V}_1$ e $\mathcal{V}_2$. As assertivas de $\mathcal{V}_2$ com Autores_Exp são obtidas diretamente das assertivas do TAutor[$\mathcal{V}_2$] com TAutor. Figura 4.24 mostra a consulta XQuery Q gerada automaticamente a partir das assertivas de $\mathcal{V}_2$.

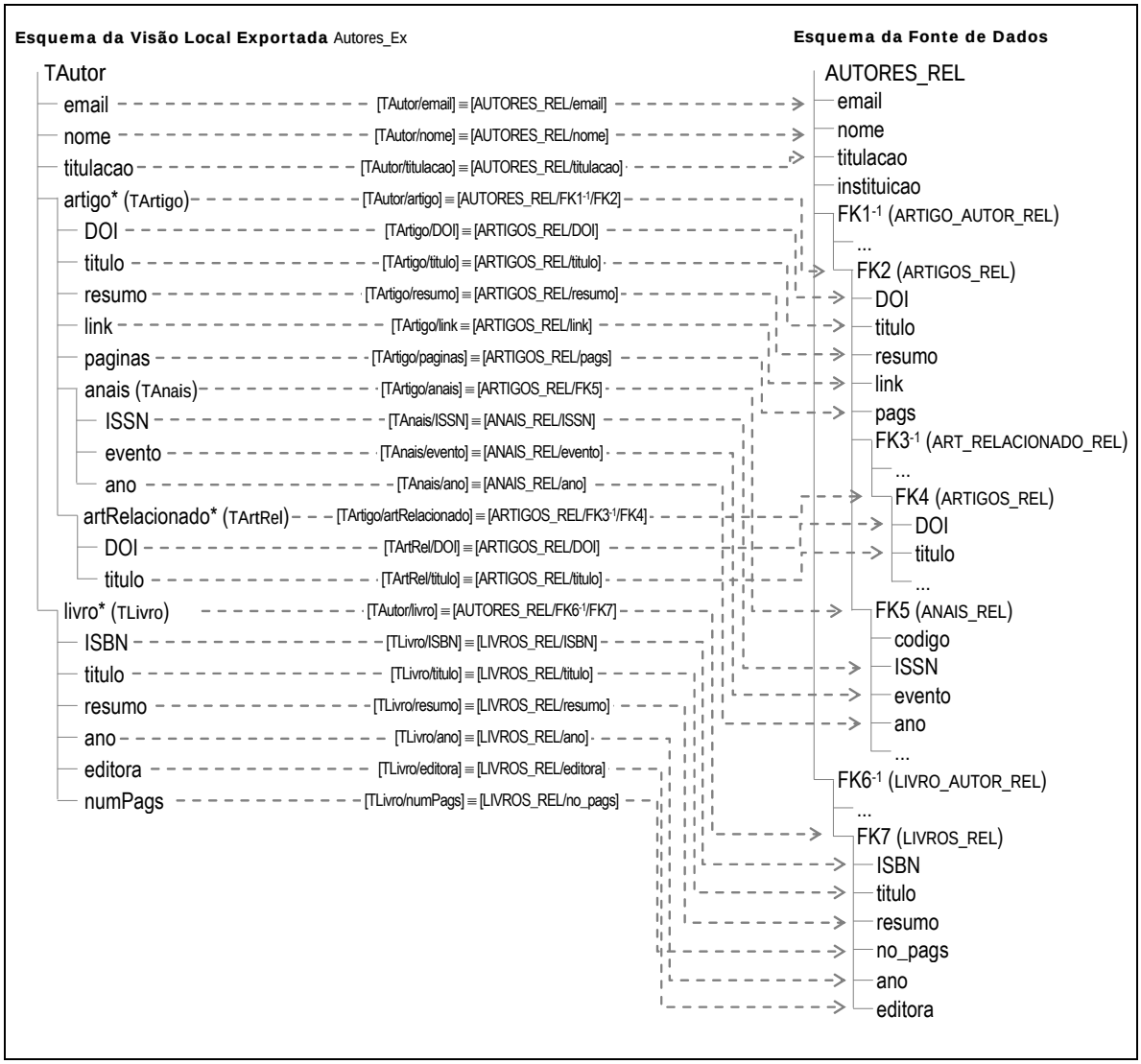

Figura 4.23. Assertivas da visão local exportada Autores_Ex

```
for $au in view("Autores_Ex")/autor
where $au/email = @email
return
   <autor>{
      $au/email, $au/nome, $au/titulacao,
      for $ar in $au/artigo
      return
         <artigo>{
            $ar/DOI, $ar/titulo,  $ar/resumo, $ar/ano
         }</artigo>,
      for $li in $au/livro
      return
         <livro>{
            $li/ISBN, $li/titulo,  $li/resumo, $li/ano
         }</livro>
   }</autor>
```

Figura 4.24. Consulta XQuery Q gerada a partir das assertivas de $\mathcal{V}_2$

4.4 Visões de Navegação Definidas sobre Múltiplas Bases

Nesta seção, é proposto um processo para sistematizar as tarefas de especificação, implementação e manutenção das VNs para aplicações DIWA que extraem seus dados de múltiplas fontes de dados.

O processo proposto também consiste de duas fases (Projeto Conceitual e Projeto Navegacional) e também parte dos cenários e casos de uso da aplicação para obter as VNs. Os cenários e casos de uso podem ser obtidos do mesmo modo como discutido na Seção 2.

Suponha agora que a aplicação *WebBib* apresentada na Seção 2 integre informações das fontes F_1, F_2 e F_3, cujos esquemas são mostrados na Figura 4.25. No esquema da fonte F_2, ℓ:*ANAIS_REL2* $\xrightarrow[f]{}$ *PERIODICOS_REL* é uma ligação entre bases distintas, onde *f*:*ANAIS_REL2*{ISSN} $\rightarrow$ *PERIODICOS_REL*{ISSN}. Para essa aplicação, considere a tarefa "*Consultar publicações de um autor*" da Seção 2 (Figura 4.9).

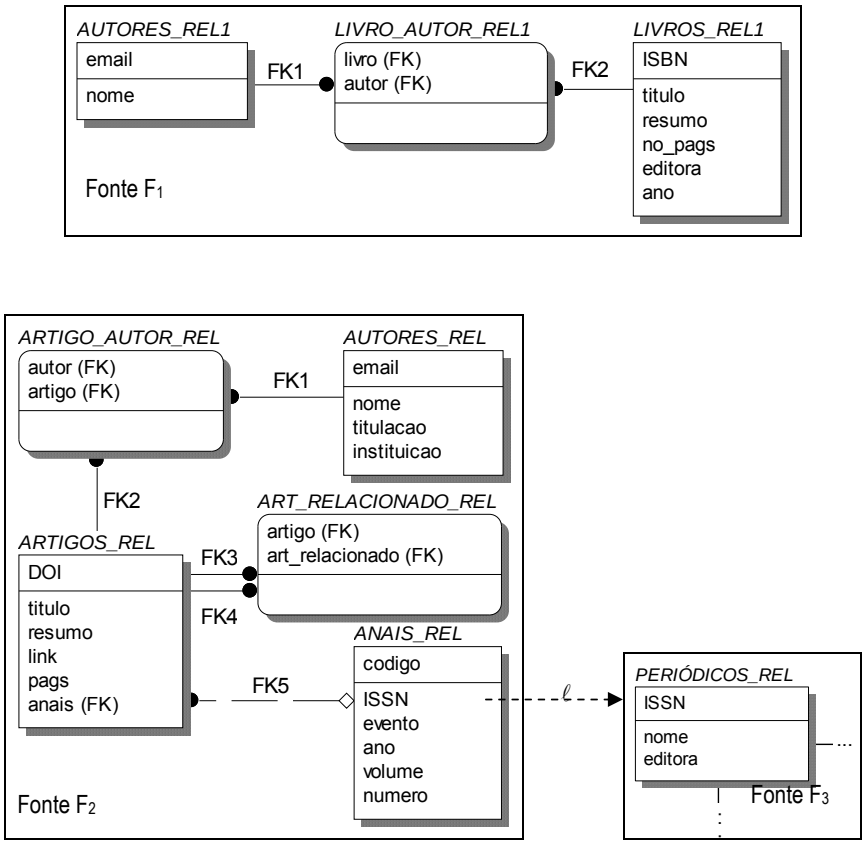

Figura 4.25. Esquema das Fontes Locais F_1, F_2 e F_3

4.4.1 Projeto Conceitual

O Projeto Conceitual é realizado em três passos:

1. Modelagem dos Esquemas das Visões de Navegação
2. Geração do Esquema Conceitual da Aplicação
3. Geração do Mapeamento entre o ECA e os Esquemas das Fontes Locais

Os passos 1 e 2 nesse processo são idênticos aos passos 1 e 2 do Projeto Conceitual da Seção 2. O que difere é que o ECA é especificado em termos de mais de uma fonte de dados. Deste modo, discutiremos a seguir apenas o Passo 3.

Passo 3: Geração do Mapeamento entre o ECA e os Esquemas das Fontes Locais

As assertivas do ECA são obtidas através do *matching* do ECA com os esquemas das fontes locais. Este problema está sendo tratado em outro trabalho [42]. As ACs do ECA com os esquemas das fontes locais são mostrados na Figura 4.26.

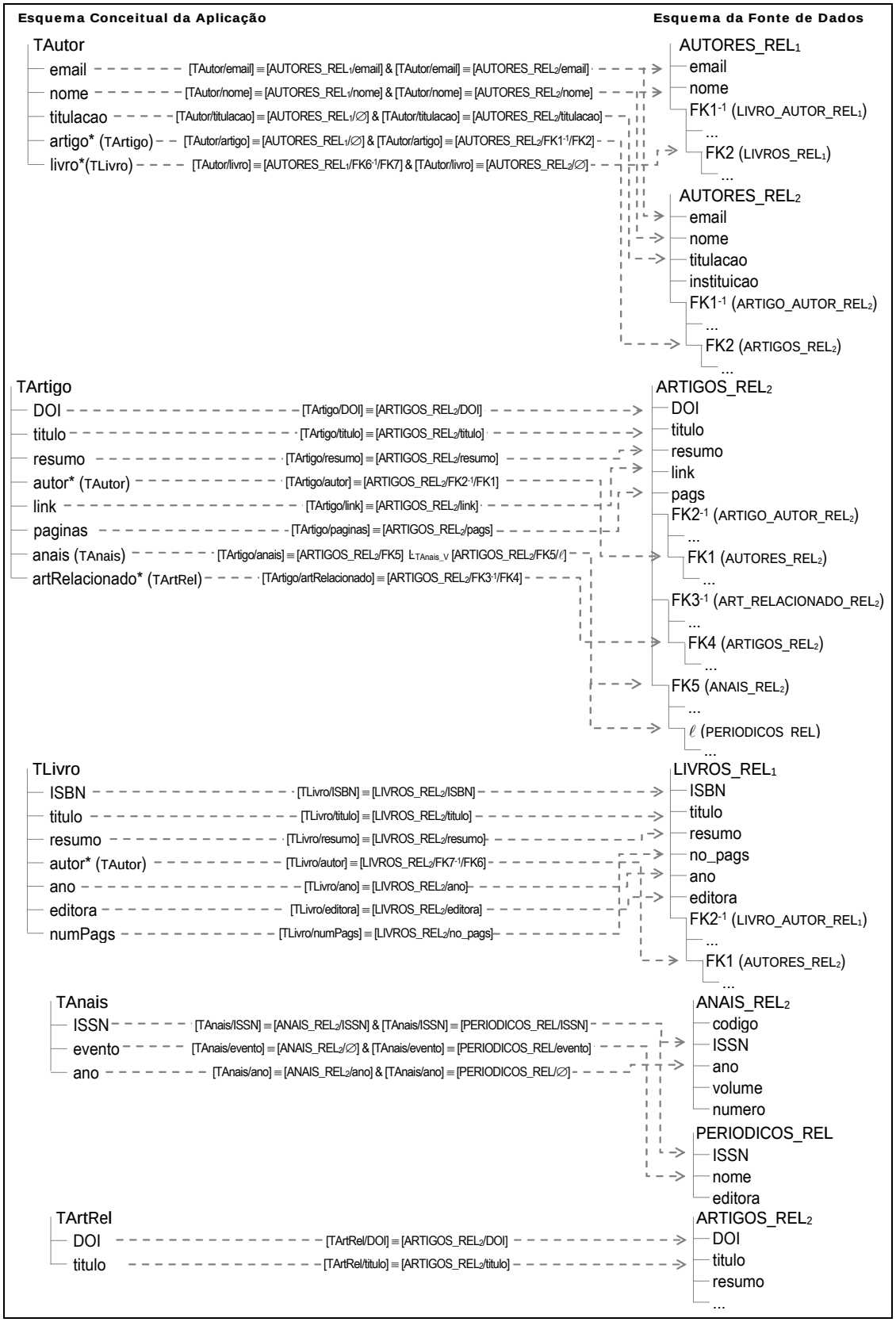

Figura 4.26. Assertivas de Correspondência do ECA com as fontes locais

4.4.2 Projeto Lógico

Na fase do Projeto Lógico consideramos duas abordagens de implementação: com e sem o uso de um mediador. Os mediadores têm como objetivo prover uma interface uniforme para consultas que requerem extração e combinação de dados de múltiplas fontes heterogêneas e autônomas [24]. Deste modo, as consultas ao mediador são formuladas em termos de visões de mediação, as quais são visões de integração definidas sobre as fontes locais. A reformulação de consultas sobre uma visão de mediação é feita de forma transparente para o usuário, ou seja, dada uma consulta sobre uma visão de mediação, o mediador localiza as fontes de dados relevantes para a consulta, reescreve essa consulta em consultas sobre as fontes locais, combina os resultados e apresenta o resultado final ao usuário. Nesse caso, as VNs são implementadas como uma única consulta XQuery sobre o esquema do mediador, cabendo ao mediador gerar, em tempo de execução, o plano de materialização da visão de navegação a partir da consulta XQuery. Ou seja, o mediador reformula a consulta XQuery da VN no plano de materialização equivalente.

Na implementação sem uso de mediador, os planos de materialização de cada VN devem ser gerados em tempo de projeto baseado nas assertivas da VN com as fontes de dados.

A vantagem da solução com mediador é que a consulta que define uma VN é em geral mais simples que na solução sem o uso de um mediador, e torna ainda mais fácil a manutenção das VNs no caso de modificações nos esquemas locais [42]. No entanto, os mediadores disponíveis atualmente são sistemas complexos, caros e que não garantem eficiência no processamento das consultas. O uso de mediadores é mais aconselhável para aplicações *Web* onde as consultas são *ad-hoc*.

4.4.2.1 Projeto Lógico com uso de um Mediador

Esta solução se aplica no caso em que um Sistema de Integração é disponibilizado. Neste caso, são criadas no mediador visões XML definidas de acordo com o ECA: para cada tipo do ECA que é tipo base de uma VN, deve-se criar no mediador uma visão XML de estrutura compatível com esse tipo. Denominamos *visão de mediação base da VN* a visão exportada pelo mediador cujo tipo é compatível com o tipo base da VN.

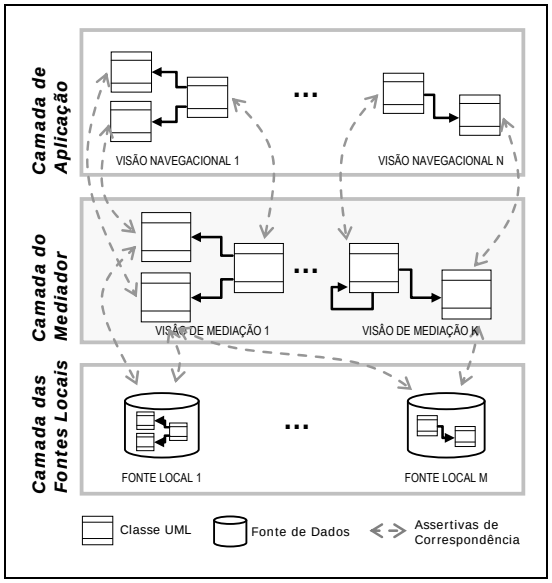

Figura 4.27. Arquitetura de Esquemas da Solução (iii)

Deste modo, uma VN é especificada sobre o esquema de sua visão de mediação base, como mostra a . As assertivas da VN com a visão de mediação base são obtidas das ACs da VN com o ECA. Nesta solução, a implementação da VN é bem simples, pois consiste de uma única consulta *XQuery*, gerada automaticamente a partir das ACs da VN com o ECA. Fica a cargo do mediador a geração do plano de materialização da consulta em tempo de execução [42]. O plano de materialização gerado pelo mediador deve ser semelhante ao obtido pela abordagem sem uso de um mediador, apresentada na Seção 4.4.2.2.

No caso da VN $\mathcal{V}_2$, é gerada a visão de mediação Autores_VM sobre as fontes F_1 e F_2. Autores_VM têm estrutura compatível com o tipo base de $\mathcal{V}_2$: TAutor. As assertivas de Autores_VM com as fontes, mostrada na Figura 4.28, é derivada das assertivas do ECA com as fontes. A consulta XQuery que define $\mathcal{V}_2$ é mostrada na Figura 4.29.

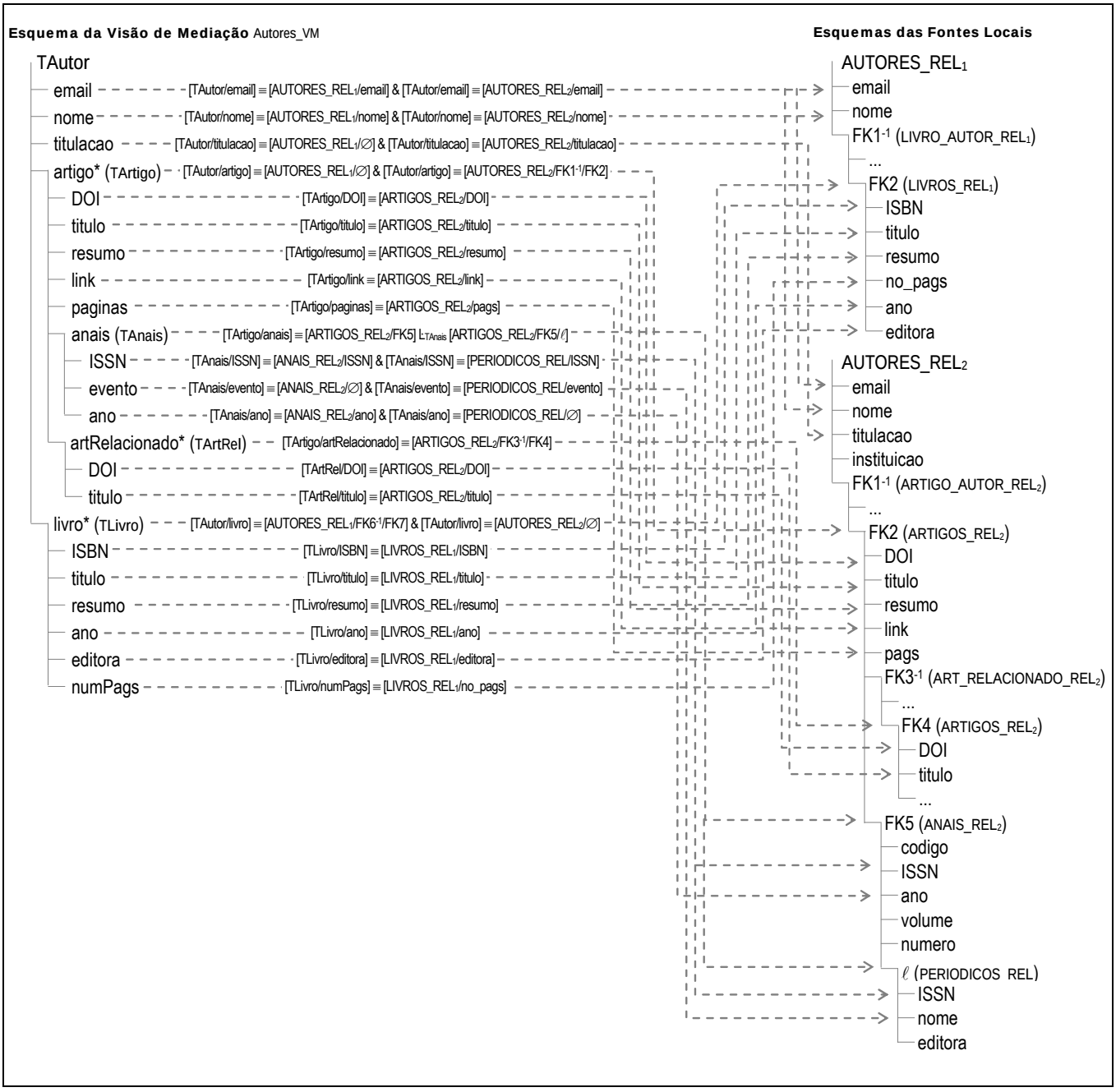

Figura 4.28. Assertivas da visão de mediação Autores_VM

```
for $au in view("Autores_VM")/autor
where $au/email = @email
return
   <autor>{
      $au/email, $au/nome, $au/titulacao,
      for $ar in $au/artigo
      return
         <artigo>{
            $ar/DOI, $ar/titulo, $ar/resumo, $ar/ano
         }</artigo>,
      for $li in $au/livro
      return
         <livro>{
            $li/ISBN, $li/titulo, $li/resumo, $li/ano
         }</livro>
   }</autor>
```

Figura 4.29. Consulta XQuery Q sobre o esquema da visão de mediação Autores_VM

4.4.2.2 Projeto Lógico sem uso de um Mediador

Neste caso, o plano de materialização da VN é gerado em tempo de projeto. Consideramos duas soluções para a geração do plano de materialização:

- Solução (i) - *Consultas às fontes locais são definidas sobre os esquemas locais*. Como mostrado na , as ACs da VN especificam o mapeamento entre o esquema da VN e os esquemas das fontes locais. Deste modo, as consultas do plano de materialização da VN são definidas sobre o esquema das fontes locais.
- Solução (ii) - *Consultas às fontes locais são definidas sobre as visões locais exportadas*. Como mostra a , as fontes exportam visões XML (Visões Locais Exportadas), cujos tipos são compatíveis com o ECA e cujas assertivas são obtidas das assertivas do ECA com as fontes. As ACs da VN especificam o mapeamento entre o esquema da VN e os esquemas das visões XML locais exportadas pelas fontes. Deste modo, as consultas do plano de materialização da VN são definidas sobre o esquema das visões locais exportadas. As consultas às fontes locais são definidas em *XQuery*, sendo estas consultas bem simples, uma vez que os esquemas das visões exportadas são compatíveis com os esquemas das VN. Outra vantagem é que mudanças no esquema das bases locais, não afetam as VNs, tornando a aplicação de fácil manutenção.

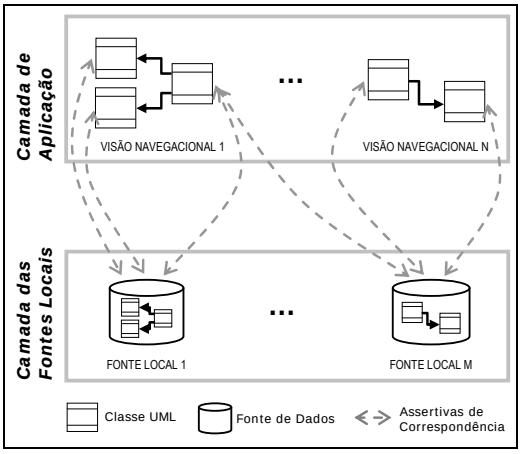

Figura 4.30. Arquitetura de Esquemas da Solução (i)

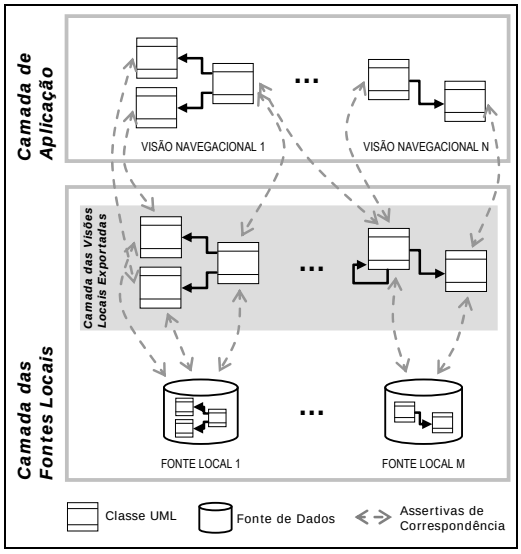

Figura 4.31. Arquitetura de Esquemas da Solução (ii)

Solução (i): *Consultas às fontes locais são definidas sobre os esquemas locais*

Neste caso, as ACs da VN especificam o mapeamento entre o esquema da VN e os esquemas das fontes locais, e são derivadas das ACs da VN com o ECA e das ACs do ECA com esquemas das fontes locais.

O plano de materialização da VN é gerado automaticamente a partir das ACs da VN, como descrito na Seção 3.5. No caso de fontes relacionais ou objeto-relacionais, as consultas do plano são definidas em SQL/XML. No caso de fontes XML, as consultas do plano são definidas em *XQuery*.

Figura 4.32 mostra as assertivas da VN $\mathcal{V}_2$ com as fontes locais. O plano de materialização de $\mathcal{V}_2$ é mostrado na Figura 4.33. No plano, o processo PP[*AUTORES_REL*$_1$] é um processo simples que realiza a consulta SQL/XML Q_1 (Figura 4.34) sobre a relação AUTORES_REL$_1$. O processo PP[*AUTORES_REL*$_2$] também é um processo simples que realiza a consulta SQL/XML Q_2 (Figura 4.35) sobre a relação AUTORES_REL$_2$. As consultas do plano são geradas automaticamente a partir das assertivas pelo algoritmo GeraConsultaSQL/XML.

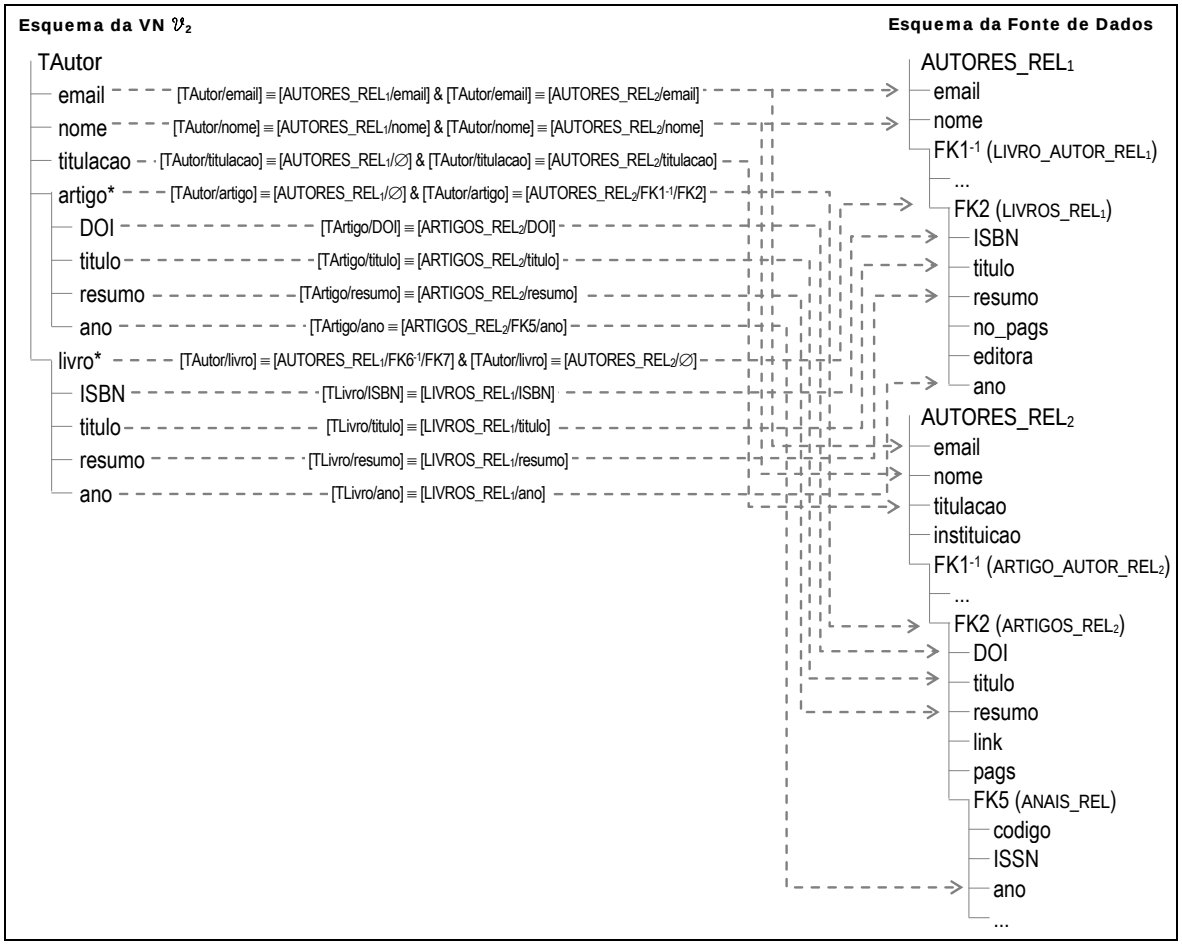

Figura 4.32. Assertivas de $\mathcal{V}_2$ com as fontes locais

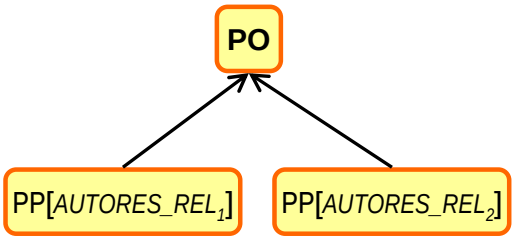

Figura 4.33. Plano de Materialização da VN $\mathcal{V}_2$ para a Solução (i)

```
SELECT XMLElement("autor",
  XMLForest("nome", AU.nome),
  XMLForest("email", AU.email),
  (SELECT XMLAgg( XMLElement("livro",
     XMLForest("ISBN", LI.ISBN),
     XMLForest("titulo", LI.titulo),
     XMLForest("resumo", LI.resumo),
     XMLForest("ano", LI.ano ) ) )
   FROM LIVRO_AUTOR_REL1 LA, LIVROS_REL1 LI
   WHERE LA.autor = AU.email AND LA.livro = LI.ISBN) )
FROM AUTORES_REL1 AU
WHERE AU.email = '@email'
```

Figura 4.34. Consulta SQL/XML Q_1 gerada a partir das assertivas de $\mathcal{V}_2$ com a fonte local F_1

```
SELECT XMLElement("autor",
  XMLForest("nome", AU.nome),
  XMLForest("email", AU.email),
  XMLForest("titulacao", AU.titulacao),
  (SELECT XMLAgg( XMLElement("artigo",
     XMLForest("DOI", AR.DOI),
     XMLForest("titulo", AR.titulo),
     XMLForest("resumo", AR.resumo),
     XMLForest("ano", (SELECT AN.ano FROM ANAIS_REL AN
                       WHERE AR.anais = AN.codigo) ) ) )
   FROM ARTIGO_AUTOR_REL2 AA, ARTIGOS_REL2 AR
   WHERE AA.autor = AU.email AND AA.artigo = AR.DOI) )
FROM AUTORES_REL AU
WHERE AU.email = '@email'
```

Figura 4.35. Consulta SQL/XML Q_2 gerada a partir das assertivas de $\mathcal{V}_2$ com a fonte local F_2

Solução (ii): *Consultas às fontes locais são definidas sobre as visões locais exportadas*

Neste caso, as fontes exportam visões XML (Visões Locais Exportadas), cujos tipos são compatíveis com o ECA e cujas assertivas são derivadas das assertivas do ECA com as fontes. No caso de fonte relacional ou objeto-relacional que possui mecanismo de criação de visões SQL/XML, as visões exportadas são geradas automaticamente a partir das ACs do ECA com a fonte de dados, de acordo com o algoritmo proposto na Seção 2.7. Denominamos *visões base da* VN as visões locais exportadas cujos tipos são compatíveis com o tipo base da VN. Note que uma visão local exportada pode ser visão base de mais de um VN.

Neste caso, as ACs de uma VN especificam o mapeamento entre o esquema da VN e os esquemas das visões XML locais exportadas pelas fontes, como mostra a . Desta forma, fica a cargo do tradutor da fonte resolver o problema de heterogeneidade estrutural [45], caso exista.

As assertivas de uma VN com os esquemas das visões exportadas são derivadas das ACs da VN com o ECA e das ACs do ECA com os esquemas das visões exportadas. O plano de materialização da VN é gerado automaticamente a partir das ACs da VN, como descrito na Seção 3.5. As consultas do plano são definidas em XQuery.

No caso da VN $\mathcal{V}_2$, são geradas as VLEs Autores_EX$_1$, Autores_EX$_2$ e Anais_Ex, sobre as fontes F_1, F_2 e F_3, respectivamente. Autores_EX$_1$ e Autores_EX$_2$ têm estrutura compatível com o tipo base de $\mathcal{V}_2$ (TAutor) e Anais_Ex têm estrutura compatível com o tipo TAnais. As assertivas de Autores_EX$_1$, Autores_EX$_2$ e Anais_Ex, geradas pelo algoritmo GeraVLEs, são mostradas nas Figuras 4.36, 4.37 e 4.38, respectivamente.

O plano de materialização de $\mathcal{V}_2$ é mostrado na Figura 4.39. No plano, o processo PP[Autores_Ex$_1$] é um processo simples que realiza a consulta XQuery Q_1 (Figura 4.40) sobre a VLE Autores_Ex$_1$. O processo PP[Autores_Ex$_2$] também é um processo simples que realiza a consulta XQuery Q_2 (Figura 4.41) sobre a VLE Autores_Ex$_2$. As consultas do plano são geradas automaticamente a partir das assertivas pelo algoritmo GeraConsultaXQuery.

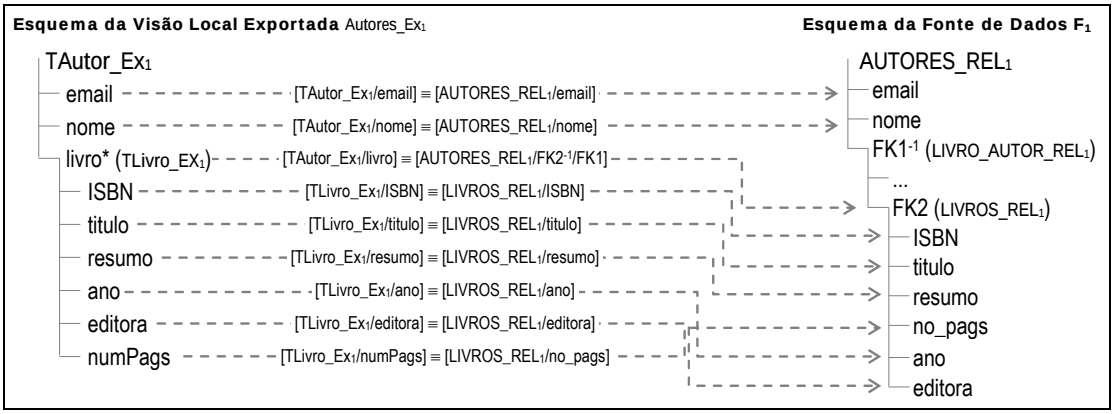

Figura 4.36. Assertivas da visão local exportada Autores_Ex$_1$ sobre a fonte F_1

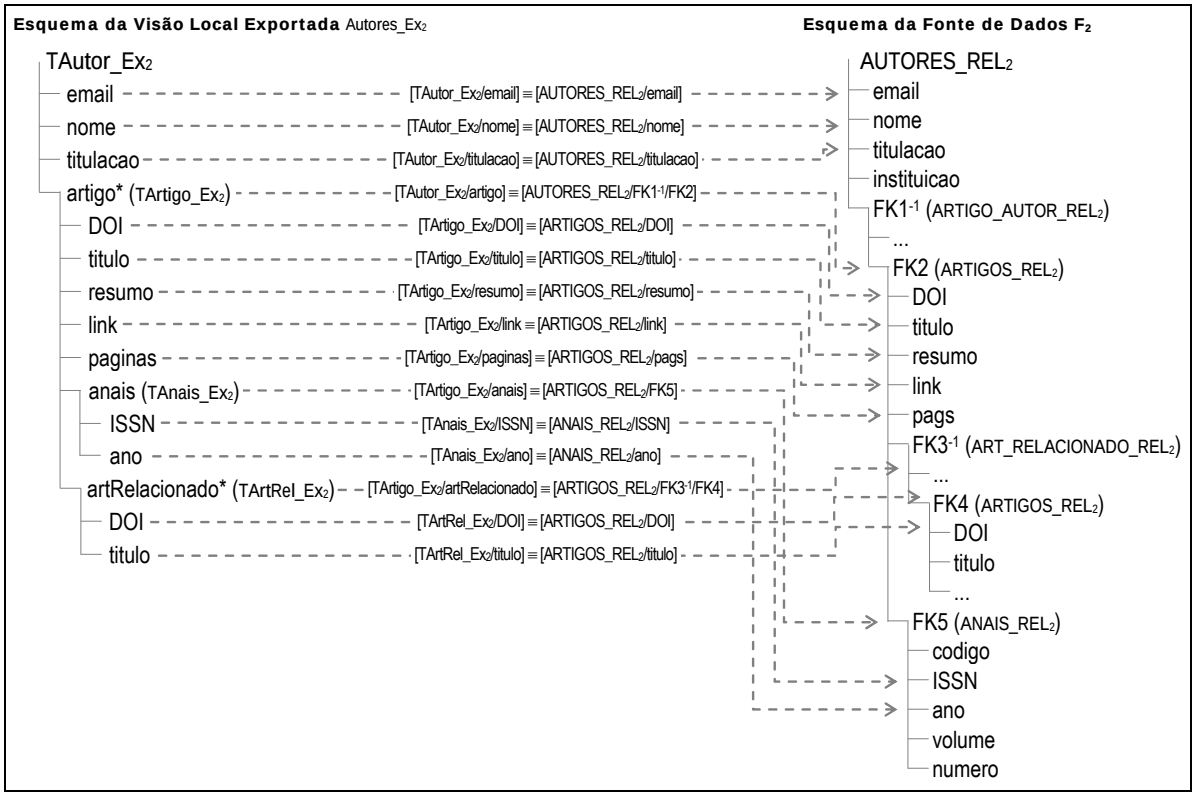

Figura 4.37. Assertivas da visão local exportada Autores_Ex$_2$ sobre a fonte F_2

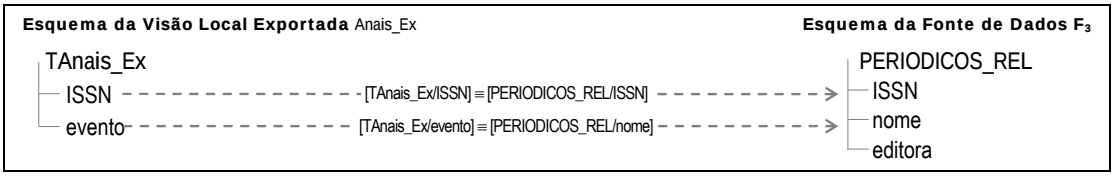

Figura 4.38. Assertivas da visão local exportada Anais_Ex sobre a fonte F_3

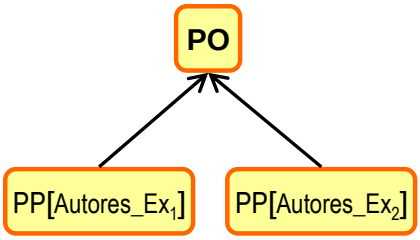

Figura 4.39. Plano de Materialização da VN $\mathcal{V}_2$ para a Solução (ii)

```
for $au in view("Autores_Ex1")/autor
where $au/email = @email
return
  <autor>{
    $au/email, $au/nome
    for $li in $au/livro
    return
      <livro>{
        $li/ISBN, $li/titulo,
        $li/resumo, $li/ano
      }</livro>
  }</autor>
```

Figura 4.40. Consulta XQuery Q_1 gerada a partir das assertivas de $\mathcal{V}_2$ com Autores_Ex1

```
for $au in view("Autores_Ex2")/autor
where $au/email = @email
return
  <autor>{
    $au/email, $au/nome, $au/titulacao,
    for $ar in $au/artigo
    return
      <artigo>{
        $ar/DOI, $ar/titulo,
        $ar/resumo, $ar/ano
      }</artigo>
  }</autor>
```

Figura 4.41. Consulta XQuery Q_2 gerada a partir das assertivas de $\mathcal{V}_2$ com Autores_Ex2

4.5 Algoritmo GeraVLEs

Nesta seção, apresentamos o algoritmo GeraVLEs, o qual é usado na geração das visões locais exportadas, como visto nas Seções 4.3 e 4.4.

O algoritmo GeraVLEs mostrado na Figura 4.1 gera, a partir das assertivas de correspondência de um tipo do ECA com uma ou mais fontes, um conjunto de VLEs primárias e secundárias sobre as fontes

ENTRADA: rótulo *r*;. coleções C_1, ..., C_n, tais que T_i é o tipo de C_i, $1 \le i \le n$; conjunto $\mathcal{A}$ de assertivas que especificam completamente um tipo T em termos dos tipos T_1, ..., T_n
visão de integração de dados $V=<S, \psi, \mathcal{A}>$, onde $S=<e,T>$
SAÍDA: conjunto $\mathcal{S}_{VLEs}$ de VLEs primárias e secundárias de V

$\mathcal{S}_{VLEs} := \emptyset$;
Para cada esquema de coleção $C_i = <e_i, T_i>$, $1 \le i \le k+m+n$ Faça
  $<T_{E[Ci]}, \mathcal{A}_{E[Ci]}> :=$ GeraTipoEAssertivasDaVLE($\mathcal{A}_{[T\&T_i]}$);
  $\psi_{E[Ci]} := [\ V_{E[Ci]}\ ] \equiv [\ C_i\ ]$;
  Crie a visão $V_{E[Ci]} = <e, T_{E[Ci]}, \psi_{E[Ci]}, \mathcal{A}_{E[Ci]}>$;
  $\mathcal{S}_{VLEs} := \mathcal{S}_{VLEs} \cup \{ V_{E[Ci]} \}$.
  $\mathcal{S}_{VLEs} := \mathcal{S}_{VLEs} \cup$ GeraVLEsSecundárias(C_i, $\mathcal{A}_{[T\&T_i]}$);
Fim Para
Retorne $\mathcal{S}_{VLEs}$;

Figura 4.42. Algoritmo GeraVLEs

ENTRADA: esquema de coleção XML $C_P=<e_P,T_P>$; conjunto $\mathcal{A}_{[T\&T_P]}$ de assertivas de correspondência que especificam completamente T em termos de T_P
SAÍDA: conjunto $\mathcal{S}_{VLEs}$ de VLEs secundárias

$\mathcal{S}_{VLEs} := \emptyset$;
Para cada assertiva $\varphi:[T/e] \equiv [T_P/\delta] \text{ L}_{Te} [T_P/\delta/\ell]$, onde $\ell:C(\lambda) \xrightarrow[f]{} C_S$ e $C_S=<e_S,T_S>$, Faça
  $<T_{E[C_S]}, \mathcal{A}_{E[C_S]}> :=$ GeraTipoEAssertivasDaVLE($\mathcal{A}_{[T\&T_P]}$);
  $\psi_{E[C_S]} := [\ V_{E[C_S]}\ ] \equiv [\ C_S\ ]$;
  Crie a visão $V_{E[C_S]} = <e, T_{E[C_S]}, \psi_{E[C_S]}, \mathcal{A}_{E[C_S]}>$;
  $\mathcal{S}_{VLEs} := \mathcal{S}_{VLEs} \cup \{ V_{E[C_S]} \}$.
  $\mathcal{S}_{VLEs} := \mathcal{S}_{VLEs} \cup$ GeraVLEsSecundárias(C_S, $\mathcal{A}_{[Te\&T_S]}$);
Fim Para
Retorne $\mathcal{S}_{VLEs}$;

Figura 4.43. Algoritmo GeraVLEsSecundárias

ENTRADA: conjunto $\mathcal{A}_{[T_1\&T_2]}$ de assertivas de caminho que especifica completamente T_1 em termos de T_2
SAÍDA: tupla <T', $\mathcal{A}_{[T'\&T_2]}$>, onde T' é o tipo restrito a partir de $\mathcal{A}_{[T_1\&T_2]}$ e $\mathcal{A}_{[T'\&T_2]}$ é o conjunto de assertivas de T' obtido a partir de $\mathcal{A}_{[T_1\&T_2]}$

```
Seja A[T'&T2] := ∅;
Para cada assertiva φ de A Faça
    Se φ:[T1/e] ≡ [T2/δ], onde e é de tipo simples Te e δ ≠ ∅ Então
        AdicionaPropriedade(T', e, Te);
        A[T'&T2] := A[T'&T2] U {φ};
    Fim se
    Se φ:[T1/e] ≡ [T2/δ], onde e é de tipo complexo Te e δ ≠ ∅ Então
        <T'e, A[T'e&Tδ]> := GeraTipoEAssertivasDaVLE(A[Te&Tδ]);
        φ' := [T'/e] ≡ [T2/δ] ∈ A[T'&T2];
        AdicionaPropriedade(T', e, T'e);
        A[T'&T2] := A[T'&T2] U A[T'e&Tδ] U {φ};
    Fim se
    Se φ:[T1/e] ≡ [T2/δ] ⊑Te [T2/δ/ℓ], onde e é de tipo complexo Te Então
        <T'e, A[T'e&Tδ]> := GeraTipoEAssertivasDaVLE(A[Te&Tδ]);
        φ' := [T'/e] ≡ [T2/δ] ∈ A[T'&T2];
        AdicionaPropriedade(T', e, T'e);
        A[T'&T2] := A[T'&T2] U A[T'e&Tδ] U {φ};
    Fim se
Fim Para
Retorne <T', A[T'&T2]>;
```

Nota
O método AdicionaPropriedade(T', e, T_e) adiciona a propriedade e com tipo T_e ao tipo T'.

Figura 4.44. Algoritmo GeraTipoEAssertivasDaVLE

4.6 Conclusões

Neste capítulo, propomos um processo para apoiar projeto, implementação e manutenção de VNs para aplicações DIWA.

No caso em que a aplicação extrai dados de uma única base, as VNs são definidas através de assertivas de correspondência sobre a fonte local, podendo o projetista optar por gerar visões locais exportadas. A consulta que define a VN é gerada automaticamente a partir das assertivas da VN. Como vimos, as VLEs funcionam como uma camada lógica entre as VNs e a fonte, de forma que alterações no esquema da fonte não alteram a definição das VNs. No entanto, atualmente, poucos bancos relacionais possuem mecanismos de visão XML, o que torna a implementação das VNs como consultas SQL/XML uma solução mais genérica.

No caso em que a aplicação extrai dados de várias bases, consideramos duas abordagens: com e sem o uso de um mediador. No caso em que um mediador é utilizado, as VNs são especificadas sobre visões criadas no mediador e as consultas que

definem as VNs são geradas automaticamente a partir das assertivas das VNs com o mediador. No caso em que um mediador não é utilizado, as VNs são definidas através de assertivas de correspondência sobres as fontes locais, podendo o projetista optar por gerar visões locais exportadas. O plano de materialização da VN é gerado automaticamente a partir das assertivas da VN.

Como vimos, a vantagem do uso de um mediador é que a consulta que define a VN é bem simples, ficando a cargo do mediador resolver o problema de integração de dados. Outra vantagem é que mudanças no esquema das bases locais, não afetam as VNs, tornando a aplicação de fácil manutenção. No entanto, os mediadores disponíveis atualmente são sistemas complexos, caros e que não garantem eficiência no processamento das consultas. O uso de mediadores é mais aconselhável para aplicações *Web* onde as consultas são *ad-hoc*.

CAPÍTULO 5
Conclusões

Neste trabalho, propomos um processo para especificação e geração de VNs para aplicações *Web* cujo conteúdo é extraído de uma ou mais fontes de dados. As VNs definem os requisitos de conteúdo dinâmico de cada página de uma aplicação *Web*, e são especificadas conceitualmente através de um conjunto de assertivas de correspondência. Como vimos, uma vantagem do processo proposto é que as VNs são especificadas conceitualmente através do formalismo da assertivas, o qual é independente de tecnologia. Deste modo, a geração e manutenção das definições das VNs pode ser realizada de forma automática.

No caso de visões de navegação definidas sobre uma base relacional, estas são especificadas usando o formalismo das assertivas apresentado no Capítulo 2. No formalismo apresentado, utilizamos assertivas de correspondência para definir o mapeamento entre uma visão XML e um esquema relacional. Definimos formalmente as condições sob as quais um conjunto de assertivas de correspondência especifica completamente uma visão XML em termos do esquema base e, no caso de base relacional, mostramos que os mapeamentos definidos pelas assertivas da visão podem ser expressos como uma consulta SQL/XML. Além disso, apresentamos um algoritmo que, baseado nas assertivas de correspondência da visão, gera a consulta SQL/XML.

No caso em que as VNs são definidas sobre múltiplas bases, usamos o formalismo proposto no Capítulo 3 para especificar a visão. Tal formalismo é uma extensão do formalismo usado para especificar visões XML sobre uma única base. Também apresentamos um algoritmo para gerar, a partir das assertivas de uma visão de integração de dados, o plano de materialização da visão, o qual computa o estado da visão a partir dos estados das fontes locais.

No Capítulo 4, detalhamos um processo para especificação e geração das VNs para aplicações *Web* com base nos formalismos dos Capítulos 2 e 3. O processo proposto consiste de duas fases: *Projeto Conceitual* e *Projeto Lógico*. No *Projeto Conceitual* são obtidos os esquemas conceituais das VNs, o Esquema Conceitual da Aplicação e os mapeamentos entre as VNs e o ECA e entre o ECA e o esquema da fonte

local (ou esquemas das fontes locais). No *Projeto Navegacional*, as consultas ou plano de materialização que extraem o conteúdo das VNs são gerados automaticamente a partir de suas assertivas de correspondência (mapeamentos).

As atividades dos processos propostos no Capítulo 4 podem ser incorporadas aos métodos de desenvolvimento de software para *Web* já existentes, como o OOHDM, por exemplo. Esses métodos tratam de forma abrangente o projeto conceitual, navegacional e de interface, e abordam como pode ser feita a implementação da aplicação. No entanto, nenhum deles trata com rigor o problema da especificação, geração e manutenção das VNs. Sem um método para especificação formal das VNs, não é possível automatizar a sua implementação e manutenção.

Como trabalhos futuros, pretendemos:

(i) Implementar um ambiente para apoiar as diversas atividades dos processos propostos no Capítulo 4. Esse ambiente seria composto de várias ferramentas, as quais são classificadas em: Ferramentas de Projeto, que apóiam as atividades do Projeto Conceitual das Visões de Navegação; e Ferramentas de Implementação, que apóiam as atividades de implementação e manutenção das VNs.

(ii) Estender o formalismo das assertivas de correspondência de forma a permitir especificar novos tipos de visões XML;

(iii) Tratar visões de Navegação que permitam além de consultas, também atualizações. Neste caso, o problema refere-se à questão de traduzir atualizações definidas sobre o esquema da visão de navegação em atualizações definidas sobre as fontes locais [46].

(iv) Desenvolver um sistema de integração baseado na arquitetura de mediadores que utiliza o formalismo das assertivas para especificar os mapeamentos do esquema das visões do mediador em termos das várias fontes locais [42]. Nesse sistema, as assertivas são usadas no processo de reformulação de consultas definidas sobre as visões XML do mediador. O problema da reformulação consiste em responder de forma eficiente consultas formuladas sobre um esquema da visão, a partir de mapeamentos previamente definidos.

Referências Bibliográficas

[1] Adams, D.: *Oracle® XML DB 10g Release 2 Developer's Guide*. 2005, Disponível em: http://www.oracle.com/technology/documentation/database10gR2.html, Acessado em 23 de fevereiro de 2007.

[2] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I.: *Business Process Execution Language for Web Services version 1.1*. Technical report, 2003, Disponível em: http://www-106.ibm.com/developerworks/*Web*services/library/ws-bpel/, Acessado em 23 de fevereiro de 2007.

[3] Atzeni, P., Mecca, G., Merialdo, P.: *Design and Maintenance of Data-Intensive Web Sites*. In: Proceedings of International Conference on Extending Database Technology, EDBT98, 1998.

[4] Benham, S.E.: *IBM XML-Enabled Data Management Product Architecture and Technology*. In: XML Data Management, Native XML and XML-Enable Database Systems, Addison Wesley, 2003.

[5] Berners-Lee, T., Hendler, J., Lassila, O.: *The Semantic Web*. In: Scientific American, Maio, 2001.

[6] Carey, M. J., Kiernan, J., Shanmugasundaram, J., Shekita, E. J., Subramanian, S. N.: *XPERANTO: Middleware for Publishing Object-Relational Data as XML Documents*. In: Proceedings of the 26th International Conference on Very Large Data Bases, pp. 646–648, 2000.

[7] Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: *Designing Data-Intensive Web Applications*. Morgan Kaufmann Publishers, 2003.

[8] Conallen, J.: *Building Web Applications with UML*. Second Edition. Addison-Wesley, 2003.

[9] Eisenberg, A., Melton, J., Kulkarni, K., Michels, J.E. and Zemke, F.: *SQL:2003 has been published*. In: ACM SIGMOD Record, v. 33, n. 1 (Março 2004), COLUMN: Standards, pp. 119–126, 2004.

[10] Eisenberg, A., Melton, J.: *SQL/XML and the SQLX Informal Group of Companies*. In: ACM SIGMOD Record, v. 30, n. 3 (Setembro 2001), COLUMN: Standards, 2001.

[11] Fagin, R., Kolaitis, P. G., Miller, R. J., Popa, L.: *Data Exchange: Semantics and Query Answering*. In: Lecture Notes in Computer Science, v. 2572, Proceedings of the 9th International Conference on Database Theory, pp. 207-224, 2003.

[12] Fernández, M., Kadiyska, Y., Suciu, D., Morishima, A., Tan, W.: *SilkRoute: A framework for publishing relational data in XML*. In: ACM Transactions on Database Systems (TODS), v. 27, n. 4 (Dezembro 2002), pp. 438–493, 2002.

[13] Fernández, M., Tan, W., Suciu, D.: *Silkroute: Trading between relations and XML*. In: Proceedings of the Ninth International World Wide *Web* Conference, (WWW'9), 2000.

[14] Florescu, D., Issarny, V., Valduriez, P., Yagoub, K.: *Caching Strategies for Data-Intensive Web Sites*. In: Proceedings of the 26th International Conference on Very Large Data Bases, pp.188-199, 2000

[15] Florescu, D., Levy, A.Y., Suciu, D., Yagoub, K.: *Run-Time Management of Data Intensive Web Sites*. In: Proceedings of *Web*DB Conference, pp. 7-12, June, 1999.

[16] Fraternali, P., Paolini, P.: *Model-Driven Development of Web Applications: The AutoWeb System*. ACM Transaction on Information Systems, v. 28, n. 4, 2001.

[17] Fuxman, A., Hernandez, M. A., Ho H., Miller, R. J., Papotti, P., Popa, L.: *Nested mappings: schema mapping reloaded*. In: Proceedings of the 32nd international conference on Very Large Data Bases, pp. 67-78, 2006.

[18] Garzotto, F., Schwabe, D., Paolini, P.: *HDM – A Model Based Approach to Hypermedia Application Design*. In: ACM Transaction on Information Systems, v. 11, 1993.

[19] Güell, N., Schwabe, D., Vilain, P.: *Modeling Interactions and Navigations in Web Applications*. In: Lecture Notes in Computer Science 1921, Proceedings of the WWW and Conceptual Modeling Workshop, ER Conference, 2000.

[20] Hernandez, M.A., Miller, R.J., Haas, L., Yan, L., Ho, C.T.H., Tian, X.: *Clio: A semi-automatic tool for schema mapping*. In: International Conference on

Management of Data, Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, pp. 607, 2001.

[21] *IBM XML for Tables*. Disponível em: http://www.alphaworks.ibm.com/tech/xtable, Acessado em 23 de fevereiro de 2007.

[22] Karvounarakis, G., Christophides, V., Plexousakis, D., Alexaki, S.: *Querying RDF descriptions for community Web portals*. In: 17iemes Journees Bases de Donnees Avancees, pp. 133-144, 2001.

[23] Krishnaprasad, M., Liu, Z. H., Manikutty, A., Warner, J. W., Arora, V., Kotsolovos, S.: *Query Rewrite for XML in Oracle XML DB*. In: Proceedings of the 30th International Conference on Very Large Data Bases, pp. 1122-1133, 2004.

[24] Lenzerini, M.: *Data Integration: A Theoretical Perspective*. In: Proceedings of ACM Symposium on Principles of Database Systems, 2002.

[25] Lima, F., Schwabe, D.: *Application Modeling for the Semantic Web*. In: Proceedings of LA-*Web* 2003, IEEE Press, pp. 93-102, 2003.

[26] Lima, T. P.: *ProDIWA: um Processo Automatizável para Geração e Manutenção de Visões de Contexto de Navegação para Aplicações DIWA*. Dissertação de Mestrado, Universidade Federal do Ceará, 2004.

[27] Liu, Z. H., Krishnaprasad, M., Arora, V.: *Native XQuery Processing in Oracle XMLDB*. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 828 - 833, 2005.

[28] *Microsoft Corporation*. Disponível em: http://www.microsoft.com, Acessado em 23 de fevereiro de 2007.

[29] Muench, S.: *Building Oracle XML Applications*. O'Reilly, 2000.

[30] Murthy, R., Banerjee, S.: *XML Schemas in Oracle XML DB*. In: Proceedings of the 29th International Conference on Very Large Data Bases, pp. 1009-1018, 2003.

[31] Oracle Corporation. *Oracle XML Developer's Kit*. Disponível em: http://otn.oracle.com/tech/xml/xdkhome .html, Acessado em 23 de fevereiro de 2007.

[32] *PHP*. Disponível em: http://www.php.org, Acessado em 23 de fevereiro de 2007.

[33] Popa, L., Velegrakis, Y., Miller, R. J., Hernandez, M. A., Fagin, R., *Translating Web Data*. In: Proceedings of the International Conference on Very Large Data Bases, pp. 598–609, 2002.

[34] Rys, M.: *XML Support in Microsoft SQL Server 2000*. In: XML Data Management, Native XML and XML-Enable Database Systems, Addison Wesley, 2003.

[35] Schwabe, D., Pontes, R.A., Moura, I.: *OOHDM-Web: an environment for implementation of hypermedia applications in the WWW*. In: ACM SIG*WEB* Newsletter, v.8, n.2, pp.18-34, 1999.

[36] Schwabe, D., Rossi, G.: *An Object Oriented Approach to Web-Based Application Design*. Theory and Practice of Object Systems, Wiley and Sons, 1998.

[37] Schwabe, D., Rossi, G.: *The Object-Oriented Hypermedia Design Model*. In: Communications of the ACM, 1995.

[38] Shanmugasundaram, J., et al.: *Querying XML Views of Relational Data*. In: Proceeding of 27th VLDB Conference, pp. 261-270, 2001.

[39] Shanmugasundaram, J., et al.: *XPERANTO: Bridging Relational Technology and XML*. IBM Research Report, 2001.

[40] Sun Microsystems. *Java Servlets Technology*. Disponível em: http://java.sun.com/products/servlet/, Acessado em 23 de fevereiro de 2007.

[41] Sun Microsystems. *JavaServer Pages Technology*. Disponível em: http://java.sun.com/products/jsp/, Acessado em 23 de fevereiro de 2007.

[42] Teixeira, G.M.L.: *Um Enfoque baseado em Assertivas para Reescrita de consultas sobre visões de Integração de Dados XML*. Dissertação de Mestrado, Universidade Federal do Ceará, 2004 (em andamento).

[43] Vdovjak, R., Frasincar, F., Houben, G. J., Barna, P.: *Engineering Semantic Web Information Systems in Hera*. In: Journal of *Web* Engineering, Rinton Press, v.2, n. 1&2, pp. 03–26, 2003.

[44] Vidal, V. M. P., Araujo, V. S., Casanova, M. A.: *Towards Automatic Generation of Rules for the Incremental Maintenance of XML Views over Relational Data*. In: WISE 2005, pp. 189-202, 2005.

[45] Vidal, V.M.P, Lóscio, B.F.: *Solving the Problem of Semantic Heterogeneity in Defining Mediator Update Translators*. In: Proceedings of 18th International Conference on Conceptual Modeling, pp. 293-308, 1999.

[46] Vidal, V.M.P, Lóscio, B.F.: *Updating multiple databases through mediators*. In: Enterprise information systems, pp. 115-122, 2000

[47] Vidal, V.M.P., Casanova, M.A., Lemos, F.C.: *Automatic Generation of XML Views of Relational Data*. In: Technical Report, 2005, Disponível em: http://lia.ufc.br/~arida, Acessado em 23 de fevereiro de 2007.

[48] Vidal, V.M.P., Lemos, F.C.: *Automatic Generation of SQL/XML Views*. In: Proceedings of 21st Brazilian Symposium on Databases, pp. 221-235, 2006.

[49] Vilain, P., Schwabe, D., Sieckenius, C.A.: *Diagrammatic Tool for Representing User Interaction in UML*. In: Lecture Notes in Computer Science, Proceedings of UML'2000, York, 2000.

[50] World Wide *Web* Consortium. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation, 2004, Disponível em: http://www.w3.org/TR/rdf-concepts/, Acessado em 23 de fevereiro de 2007.

[51] World Wide *Web* Consortium. *XML Schema Part 2: Datatypes*. W3C Recomendation, 2001, Disponível em: http://www.w3.org/TR/xmlschema-2/, Acessado em 23 de fevereiro de 2007.

[52] World Wide *Web* Consortium. *XQuery 1.0: An XML Query Language*. W3C Working Draft, 2004, Disponível em: http://www.w3.org/TR/xquery/, Acessado em 23 de fevereiro de 2007.

[53] Yu, C., Popa, L.: *Constraint-Based XML Query Rewriting For Data Integration*. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of data, SESSION: Research sessions: Data Integration, pp. 371–382, 2004.