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

Dissertação de Mestrado

ProDIWA: um Processo Automatizável para Geração e Manutenção de Visões de Contexto de Navegação para Aplicações DIWA

Tâmara Peixoto Lima

Fortaleza - Ceará
2004

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

PRODIWA: UM PROCESSO AUTOMATIZÁVEL PARA GERAÇÃO E MANUTENÇÃO DE VISÕES DE CONTEXTO DE NAVEGAÇÃO PARA APLICAÇÕES DIWA

Tâmara Peixoto Lima

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

Fortaleza - Ceará
Setembro de 2004

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

PRODIWA: UM PROCESSO AUTOMATIZÁVEL PARA GERAÇÃO E MANUTENÇÃO DE VISÕES DE CONTEXTO DE NAVEGAÇÃO PARA APLICAÇÕES DIWA

Tâmara Peixoto Lima

Orientadora: Profª. Drª. Vânia Maria Ponte Vidal

Aprovada em __/__/_____

BANCA EXAMINADORA

Prof.ª Dr.ª Vânia Maria Ponte Vidal
Universidade Federal do Ceará-UFC

Prof. Dr. Daniel Schwabe
Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO

Prof. Dr. Fernando Antônio de Carvalho Gomes
Universidade Federal do Ceará - UFC

Prof. Dr. Cidcley Teixeira de Souza
Centro Federal de Educação Tecnológica do Ceará - CEFET

Agradecimentos

Agradeço principalmente a Deus, a quem eu sempre recorro e que jamais me permitiu desistir, por sua infinita bondade e generosidade.

Aos meus pais Varnô e Josefinha, por todo o amor e apoio, pelo exemplo, e por estabelecerem um ambiente propício para o meu crescimento intelectual.

Aos meus irmãos Glauton, Glaudênia e Siomara, ao meu cunhado Júlio, e minha prima Marcela, que estiveram sempre presentes nesta luta, apoiando-me e animando-me.

À minha orientadora Vânia, a quem respeito e admiro, por toda a dedicação, horas e paciência empregadas, e por estar sempre cobrando o melhor possível.

Ao professor Cidcley, pela contribuição e incentivo oferecidos e pelo seu otimismo constante.

A Fundação Cearense de Apoio ao Desenvolvimento Científico e Tecnológico – FUNCAP, órgão que fomentou a pesquisa, dando-me apoio financeiro necessário para o desenvolvimento da mesma.

Ao Rafael, pelo amor, compreensão, apoio e por tudo mais que eu precisasse.

À minha turma de mestrado, Cynthia, Desiere, Tarciane, Paulo Henrique, William, Erick, Misael, Daniel, Bringel, Carlos Sérgio, Robledo, pelos momentos de alegria, estudo e desespero compartilhados, e em especial, à Desiere, por se mostrar uma amiga de todas as horas.

Aos meus colegas do laboratório ARIDA, especialmente ao Lineu, Valdiana, Fernando e Denis, pelo incentivo e grande colaboração.

Aos professores do Mestrado em Ciência da Computação da UFC, por todo conhecimento transmitido e pelo empenho em tornar o mestrado cada vez melhor, e aos funcionários Orley e Rosana.

Às minhas amigas de sempre, Sâmia e Ana Cláudia, pela amizade e companheirismo.

Aos meus amigos da graduação, especialmente a Erika, Leonardo, Daniel, Roger e Artur, pessoas que estiveram sempre presentes durante meu curso, e que estando hoje onde estiverem, vivem no meu coração.

E a todas as pessoas que colaboraram de alguma forma com essa dissertação, meus eternos agradecimentos.

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) [8]. Neste trabalho, os requisitos de conteúdo de cada página da aplicação são especificados através de uma visão, a qual denominamos **Visão de Contexto de Navegação** (VCN). Consideramos que os dados das VCN's estão armazenados em um banco de dados relacional ou objeto-relacional.

Neste trabalho, propomos um processo para geração e manutenção das VCN's para aplicações DIWA. O processo compreende cinco atividades, a saber: Levantamento de Requisitos, Projeto das Visões de Objeto, Projeto das VCN's, Implementação e Manutenção das VCN's. Uma das vantagens do enfoque proposto é que a implementação e a manutenção das VCN's podem ser realizadas de forma automática, a partir de suas especificações conceituais. As VCNs podem ser implementadas como visões de objeto ou como visões XML.

Abstract

Web applications for accessing and maintaining large amounts of structured data, typically stored as records in a database management system, are called "data-intensive Web applications" (DIWA applications) [8]. In this work, the content requirements for each page of the application are specified through a view, denominated **Navigation Context View** (VCN). We consider that VCN's data are stored in a relational or object-relational database.

In this work we propose a process for generating and maintaining VCN's for DIWA applications. This process comprises five activities: Requirements Analysis, Conceptual Design, VCN's Design, Implementation and Maintenance of the VCN's. One advantage of our approach is that the implementation and maintenance of the VCN's can be done in an automatic way based on VCNs' conceptual specifications. We consider that the VCN's can be implemented either as object views or as XML views.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

1.1 Aplicações *Web*

A *web* é um dos meios mais utilizados para disseminação e compartilhamento de informações a nível mundial. 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 [2].

Devido a grande diversidade de *sites web*, Atzeni [2] propõe a seguinte classificação:

- *Web-presence sites*: são os *sites* que possuem baixa complexidade em termos de dados e aplicações. Esses *sites* geralmente contêm um número pequeno de páginas e servem principalmente aos propósitos de publicidade e propaganda. Eles possuem tamanho relativamente pequeno e são em geral feitos manualmente, com auxílio de editores HTML ou *softwares* gerenciadores de *site*;
- *Service-oriented sites*: são os *sites* dedicados a algum serviço específico, como por exemplo, *sites* de busca e serviço de *e-mail* grátis. Nesses *sites*, apesar de ser

necessário realizar o armazenamento de dados, a estrutura desses dados e do hipertexto é bem simples e muito específico para cada tipo de serviço;

- *Catalogue sites*: são os *sites* que publicam grande quantidade de dados, e geralmente possuem uma estrutura de hipertexto complexa, mas oferece pouco ou nenhum serviço. O principal foco é na organização dos dados de forma navegável e na manutenção dos dados armazenados e do hipertexto;
- *Web-Based Information Systems*: são os *sites* que são verdadeiros sistemas de informação na *web*, oferecendo acesso aos dados complexos e ao mesmo tempo fornecendo serviços interativos. Enquadram-se nessa categoria os grandes *sites* de comércio eletrônico e os sistemas de intranet.

A classificação de um *site* não é uma tarefa precisa, uma vez que alguns *sites* podem se enquadrar em mais de uma das categorias citadas acima. Os sites que oferecem ao usuário a possibilidade de interferir na lógica de negócios, que em geral é caso dos *catalogue sites* e *web-based information systems*, são chamados de aplicações *web* [12]. 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 [8].

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 dos *browsers* do lado cliente. Na camada servidora de dados, os dados são armazenados de forma persistente e podem ser organizados e selecionados. A camada mediadora ou 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* filtrados a partir da camada mediadora.

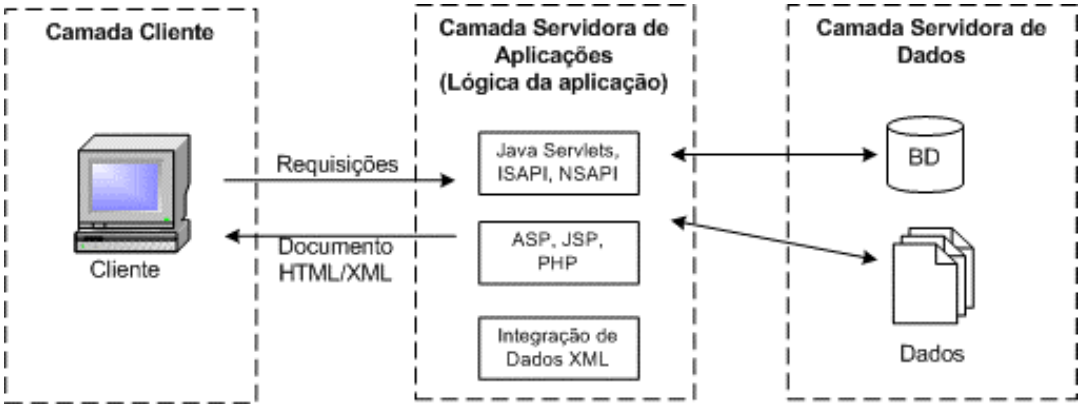


Figura 1.1: Camada 3 níveis

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 [12]. Algumas implementações mais populares dessas tecnologias são *Internet Server* API (ISAPI) [31] e Java *Servlets* [48], no caso de módulos compilados, e *JavaServer Pages* (JSP) [49], *Active Server Pages* (ASP) [31] e PHP [37], 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* [22].

1.2 Motivação e Objetivos da Dissertação

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, em aplicações *web*, deve-se considerar os requisitos adicionais de combinar uma navegação controlada pelo usuário com a facilidade de acesso aos dados e o grande volume de informações disponíveis na *web*, o que 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, denominada Visão de Contexto de Navegação (VCN). Cada interação entre o usuário e o sistema, necessária para realização de uma tarefa da aplicação, possui uma VCN 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) [50][25] 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 [8], uma vez que requer intenso acesso e atualização dos dados armazenados em bases de dados. A construção e manutenção das VCN's não são tarefas fáceis em aplicações DIWA, onde um grande número de páginas dinâmicas necessita ser gerado. Nessas aplicações, as VCN's são definidas através de uma consulta SQL, a qual especifica como os objetos da visão são sintetizados a partir dos dados do banco de dados. Definir uma consulta que realize o mapeamento de tuplas de tabelas relacionais em objetos de tipos complexos é tarefa que exige conhecimentos avançados de SQL3. Esse problema se torna complexo no caso das aplicações DIWA, dada a grande quantidade de visões que precisam

ser criadas. Manter essas visões também não é tarefa simples, uma vez que mudanças no esquema do banco de dados podem afetar uma grande quantidade de VCN's que precisam, então, ser redefinidas.

O problema de especificação, geração e manutenção das VCN's não é tratado com rigor em nenhum dos métodos de desenvolvimento de aplicações *web* conhecidos. Sem um método para especificação formal das VCN's, não é possível automatizar a sua implementação e manutenção. Uma das principais contribuições deste trabalho é a proposta de um processo para especificação, geração e manutenção das VCN's para aplicações DIWA. As VCN's são especificadas conceitualmente por meio de um conjunto de Assertivas de Correspondências (AC), as quais especificam formalmente como os objetos da VCN podem ser sintetizados a partir do banco de dados. A vantagem do uso de AC's para especificação das VCN's é permitir que a implementação e a manutenção das VCN's possam ser realizadas de forma automática, a partir de suas especificações conceituais.

1.3 Visão Geral do Processo Proposto

Neste trabalho, é proposto um processo para sistematizar as tarefas de especificação, implementação e manutenção das VCN's para aplicações DIWA. Assim como em [44][24][8], nossa abordagem é centrada no usuário, pois depende da coleta dos requisitos dos usuários para desenvolver a aplicação e para projetar o banco de dados. O processo proposto, denominado ProDIWA, compreende cinco atividades (vide Figura 1.2): Levantamento de Requisitos, Projeto das Visões de Objetos, Projeto das VCN's, Implementação das VCN's e Manutenção das VCN's. A seguir, discutimos brevemente cada uma dessas atividades.

A atividade Levantamento de requisitos consiste em especificar as tarefas desempenhadas pela aplicação, através de cenários e casos de uso. Para cada caso de uso especificado, é gerado um *Diagrama de Interação dos Usuários* (UID) que representa as interações dos usuários e a aplicação para realização de uma tarefa. Cada interação do UID

possui associada uma VCN, que especifica os requisitos de conteúdo no contexto da interação.

O projeto das visões de objeto é realizado através da integração das VCN's dos UID's, de forma que estas satisfaçam os requisitos de todas as VCN's dos UID's. Dessa forma, cada VCN possui uma visão de objeto base, da qual são selecionados os objetos da VCN. As visões de objeto são especificadas conceitualmente por meio de um conjunto de assertivas de correspondências, as quais especificam formalmente as correspondências do esquema da visão de objeto com o esquema do banco de dados. Consideramos que os dados estão armazenados em um banco de dados relacional ou objeto-relacional, e que o mesmo pode já existir, ou que deverá ser projetado especificamente para a aplicação. Neste último caso, um banco de dados deve ser projetado, de acordo com uma das abordagens apresentadas em [1][16][10], que tratam do projeto de banco de dados relacional [1][16] ou objeto-relacional [10] a partir de um modelo orientado a objetos.

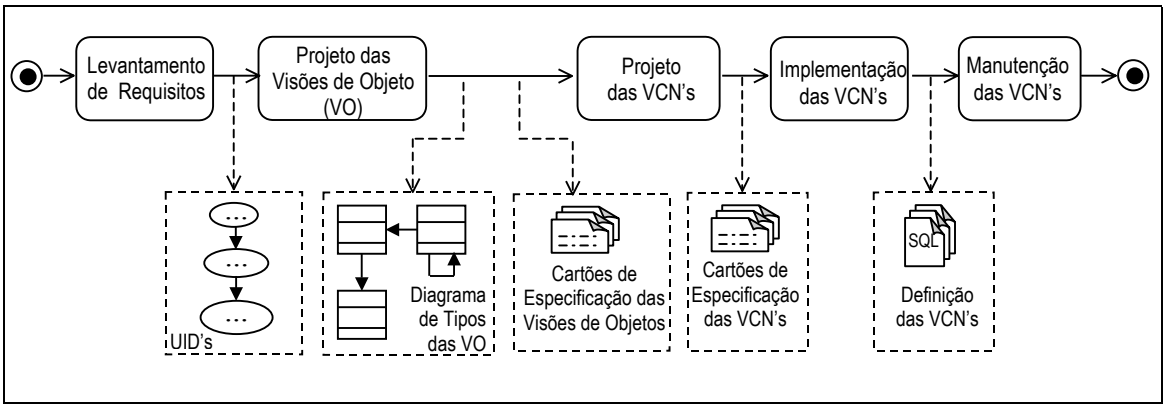


Figura 1.2: Diagrama de Atividades do Processo ProDIWA

Na atividade Projeto das VCN's, as VCN's são especificadas conceitualmente por meio de um conjunto de assertivas de correspondências as quais especificam as correspondências da VCN com o esquema da sua visão de objeto base.

Durante a implementação, cada VCN é implementada de forma automática a partir da sua especificação conceitual. Neste trabalho, as VCN's podem ser implementadas como visões de objeto ou como visões XML. O crescimento de transações business-to-business (B2B) via Internet fez surgir a necessidade das empresas por aplicações que possibilitem a troca em tempo real de informações e processos que conduzem os negócios da empresa. A

linguagem XML tem se tornado uma solução para essas necessidades. O uso de visões XML possibilita a publicação de dados armazenados em bases de dados convencionais no formato XML.

Neste trabalho, é proposto também um ambiente para apoiar as diversas atividades do ProDIWA. Esse ambiente é composto de várias ferramentas, as quais são classificadas em: Ferramentas de Projeto, que apóiam as atividades Projeto das Visões de Objetos e Projeto das VCN's; e Ferramentas de Implementação, que apóiam as atividades Implementação e Manutenção das VCN's.

1.4 Trabalhos Relacionados

Diversas metodologias surgiram com o intuito de auxiliar os projetistas a superar as dificuldades no projeto de aplicações *web* [44][45][2][3][22][23][8][24]. A maioria 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* (UID's) [50][25].

O *Hypermedia Design Method* (HDM) [23] foi um dos métodos pioneiros para tratar do projeto de aplicações hipermídia e influenciou diversas metodologias subseqüentes, como OOHDM [44][45] e HDM-lite [22]. 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* voltadas para esse tipo de aplicação. De acordo com o OOHDM, o desenvolvimento de aplicações *web* ocorre como um processo de cinco atividades: Levantamento de Requisitos, Projeto Conceitual, Projeto Navegacional, Projeto de Interface Abstrata e Implementação. Essas atividades são realizadas com uma mistura de desenvolvimento iterativo e incremental, onde em cada passo um modelo é construído ou enriquecido.

Na atividade Levantamento de Requisitos, são utilizados cenários e casos de uso para especificação das tarefas a serem realizadas pela aplicação. Além disso, UID's são utilizados para representação gráfica das interações dos usuários com o sistema para a execução de uma dada tarefa.

No Projeto Conceitual, o modelo conceitual representa o domínio dos objetos e seus relacionamentos. O esquema conceitual é construído a partir dos UID's, como mostrado em [50][25], e é representado utilizando uma notação similar a UML.

No Projeto Navegacional, o modelo navegacional é construído como uma visão sobre o modelo conceitual, permitindo a construção de diferentes modelos navegacionais a partir do mesmo modelo conceitual. O projeto navegacional é expresso 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.

No Projeto de Interface Abstrata, é estabelecido quais nós navegacionais são manifestados para os usuários e como são manifestados. 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 UID's são re-examinadas 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 [28] 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 [4].

- **Araneus**

Araneus [2][3] apresenta um processo de projeto de *sites Web* em geral, dividido em projeto de banco de dados e projeto de hipertexto. Cada uma dessas atividades é subdividida em uma fase de projeto conceitual e uma fase de projeto lógico. Durante o projeto conceitual do banco de dados, a estrutura dos dados é descrita através do Modelo Entidade-Relacionamento. O projeto conceitual do hipertexto é especificado a partir do esquema ER obtido, usando o *Navegation Conceptual Model* (NCM). Nessa fase, são definidas quais entidades do esquema conceitual representam os nós, quais os caminhos de navegação entre esses nós e quais estruturas de acesso são criadas para possibilitar o usuário explorá-los de forma hierárquica. Em seguida, o esquema lógico de hipertexto é construído usando o modelo *Araneus Design Model* (ADM). O modelo ADM representa uma descrição abstrata das páginas da aplicação. Durante essa fase, os nós são mapeados em *page-schemes*, que constituem uma descrição de um conjunto de páginas com características comuns. No projeto de apresentação, um *template* é gerado para cada *page-scheme*, o qual define a aparência das páginas que são instâncias do *page-scheme*.

Para a geração das páginas no Araneus, é necessário especificar de quais tabelas e atributos do banco de dados extrai-se o conteúdo das páginas. As correspondências entre o esquema de hipertexto (esquema ADM) e o banco de dados e a geração automática de páginas HTML a partir dessa correspondência é realizada pelo módulo Penelope. O

conteúdo de cada *page-scheme* no esquema ADM corresponde a uma visão definida por meio de uma consulta SQL sobre o banco de dados. Essa consulta SQL é definida manualmente pelo projetista. Uma vez definida a visão, é necessário associar cada atributo do *page-scheme* com o atributo correspondente na visão, usando uma linguagem apropriada. Penelope gera as páginas HTML a partir das correspondências entre o esquema ADM e o esquema do banco de dados.

Durante a geração de uma página, o módulo PENELOPE extrai os valores dos atributos do banco de dados e combina com o template da página. PENELOPE fornece duas alternativas para a geração de páginas: (i) páginas são geradas a partir do banco de dados, materializadas e armazenadas em arquivos HTML, e (ii) páginas são dinamicamente geradas a partir de uma requisição do usuário, através de módulos CGI. No caso de aplicações *data-intensive*, a utilização da alternativa (i) é desaconselhada. Devido ao grande número de páginas e intensa atualização de dados que caracteriza esse tipo de aplicação, as páginas precisariam ser freqüentemente re-geradas, o que tornaria a alternativa muito ineficiente.

- **Autoweb**

No projeto Autoweb [22], é apresentada uma metodologia para desenvolvimento de aplicações *Web*, chamada HDM-lite, que é uma versão do HDM específica para aplicações *Web*. HDM-Lite provê uma notação para especificação de estrutura, navegação e apresentação da aplicação. Além disso, Autoweb fornece uma ferramenta chamada *Autoweb System* que dá suporte à definição de aplicações *Web* utilizando HDM-lite e permite a implementação dessas aplicações através de uma abordagem *top-down*. Assim como é feito em Araneus, essa ferramenta realiza o mapeamento das estruturas do banco de dados para as páginas que constituem a aplicação. O esquema conceitual é traduzido no esquema do banco de dados, o qual descreve também as informações de estrutura das páginas, navegação e apresentação, além da organização do conteúdo. Isso viabiliza a implementação automática e a evolução da aplicação dentro do ambiente. As páginas da aplicação são geradas dinamicamente através do componente *AutoWeb Generator*. Folhas de estilo são usadas como diretivas que contêm a descrição de como produzir o conteúdo em uma linguagem de apresentação (por exemplo, HTML).

- **OO-H**

Em [24] é apresentado o método OO-H (*Object-Oriented Hypermedia Method*), que estende a UML para modelar interfaces de aplicações *Web*. De maneira similar ao OOHDM, o processo de desenvolvimento da aplicação é dividido em modelo conceitual, modelo navegacional e modelo de apresentação. O processo inicia-se com a definição de um diagrama de caso de uso e um diagrama de classes. O Diagrama de Acesso Navegacional (NAD) define o modelo navegacional, e é construído baseado nos diagramas de caso de uso e de classes. O Diagrama de Apresentação Abstrata (APD) e o Diagrama de Composição de *Layout* definem a estrutura abstrata das páginas e detalhes específicos de apresentação, respectivamente. A definição abstrata das páginas no APD é baseada em DTD's [56]. OO-H utiliza XML juntamente com folhas de estilo XSL para gerar as interfaces da aplicação.

- **WebML**

Em [8] é apresentado um processo de desenvolvimento de aplicações *Web*, composto das seguintes fases: Especificação de Requerimentos, Projeto de Dados, Projeto de Hipertexto e Desenvolvimento da Aplicação. A fase Desenvolvimento da Aplicação consiste nas seguintes atividades: Projeto de Arquitetura, Implementação dos Dados, Implementação do Hipertexto e Testes e Manutenção. Para especificação do hipertexto de aplicações DIWA, é proposta uma linguagem chamada WebML (*Web Modeling Language*). WebML possui conceitos visuais para expressar o hipertexto como um conjunto de páginas compostas por unidades de conteúdo ligadas entre si e operações e para especificar as correspondências entre tais unidades e operações com os dados aos quais essas se referem.

A linguagem WebML é baseada na noção de unidades, páginas e *links*. Unidades descrevem os pedaços de conteúdo a serem publicados. As páginas são os elementos de interfaces que são entregues aos usuários, e são compostas por diversas unidades de vários tipos. Os *links* descrevem as conexões entre unidades e páginas. Múltiplos hipertextos podem ser definidos sobre o mesmo conteúdo, para oferecer diferentes visões aos usuários.

Para apoiar as diversas fases que compõem o processo WebML, foi desenvolvida a ferramenta *WebRatio Site Development Studio* [8][52]. Essa ferramenta provê interfaces

gráficas para que o usuário possa definir o esquema entidade-relacionamento e o esquema de hipertexto, obtidos durante as fases de projeto de dados e de projeto de hipertexto, respectivamente. Além disso, *WebRatio* apóia a fase de implementação, automatizando a construção de um banco de dados relacional e dos *templates* das páginas, contendo as consultas SQL que extraem os dados de cada página. Essas consultas são geradas automaticamente a partir da definição das páginas, realizada durante o processo de hipertexto.

Algumas das metodologias apresentadas [2][22][8] surgiram com o intuito de apresentar um processo sistemático para desenvolvimento de aplicações *web*, dando maior ênfase ao caso de aplicações DIWA. 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.

A ferramenta *CodeCharge Studio* [11] provê geração automática de código baseada em *wizards* para produção de *sites web*. O projetista da aplicação pode definir um modelo do *site* em alto nível, inserindo diferentes páginas usando vários padrões e gerando automaticamente os *templates* das páginas e consultas SQL executando em diferentes plataformas.

As consultas SQL geradas pela *WebRatio* e pela *CodeCharge Studio* se restringem a consultas definidas sobre banco de dados relacionais e são consultas simples, definidas sobre uma única tabela. 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 obtidos do banco de dados em objetos de tipos complexos, onde os tipos dos objetos definidos no banco de dados e da VCN podem ter estruturas bem diferentes.

1.5 Organização da Dissertação

Os capítulos a seguir estão organizados da seguinte forma. No Capítulo 2, apresentamos o modelo objeto-relacional, que é utilizado para representar o esquema das

visões de objetos e o esquema do banco de dados. No Capítulo 3, discutimos as principais características da linguagem XML e apresentamos alguns *frameworks* existentes para publicação de dados armazenados em banco de dados relacionais e objeto-relacionais no formato XML. No Capítulo 4, abordamos as atividades de Projeto das Visões de Objetos e Projeto das VCN's. No Capítulo 5, descrevemos as atividades Implementação e Manutenção das VCN's. No Capítulo 6, apresentamos o ambiente proposto para apoiar o ProDIWA, suas ferramentas e os algoritmos utilizados nas ferramentas. No Capítulo 7, são mostradas as conclusões e sugestões para trabalhos futuros. Finalmente, no Apêndice A, apresentamos um estudo de caso para exemplificar o uso do processo proposto.

Capítulo 2
Modelo Objeto Relacional

Neste Capítulo, apresentamos o modelo objeto-relacional baseado no modelo do *Oracle* 9i [33]. Este modelo é resultado da extensão do modelo relacional para suportar os conceitos de orientação a objetos. Na Seção 2.1, descrevemos os principais elementos de modelagem que compõem o modelo Objeto-Relacional do *Oracle* 9i. Na Seção 2.2, apresentamos uma notação gráfica para representar esquemas objeto-relacionais. Na Seção 2.3, abordamos as características orientadas a objetos da linguagem SQL:1999 utilizadas nesta dissertação. Na Seção 2.4, discutindo sobre o mecanismo de Visões de Objetos implementado no *Oracle* 9i. Encerramos este Capítulo definindo os vários tipos de assertivas de correspondência, que são usadas para especificar formalmente a correspondência entre os tipos das VCN's e o esquema das visões de objeto da aplicação, e entre os esquemas dessas visões e do banco de dados.

2.1 Esquema Objeto Relacional

No SGBD *Oracle* 9i, um esquema objeto-relacional é uma tripla **S**=(**T**, **R**, **I**), onde **T** é um conjunto de definições de tipos, **R** é um conjunto de tabelas e **I** é um conjunto de restrições de integridade. Uma das principais características do modelo objeto-relacional é a capacidade de estender o sistema de tipos do banco de dados, ou seja, novos tipos podem ser definidos pelos usuários. Esses tipos são chamados de tipos abstratos de dados (TAD). Um tipo serve como molde para a criação de objetos (instâncias do tipo) e serve para definir a estrutura de dados (atributos) e as operações (métodos) que são comuns às

instâncias do tipo. Para criar um TAD em um banco de dados objeto-relacional, usamos o comando CREATE TYPE.

Os atributos de um TAD podem ser classificados como *monovalorados* ou *multivalorados*. O valor de um atributo monovalorado é um objeto (atômico ou estruturado) ou uma referência para um objeto. Já o valor de um atributo multivalorado é uma coleção de objetos ou coleção de referências para objetos. Um atributo multivalorado pode ser representado como uma *Nested Table* (coleção desordenada e ilimitada de objetos) ou *Varray* (coleção ordenada e limitada de objetos). Considere, por exemplo, o esquema objeto relacional Pedidos apresentado na Figura 2.1:

- O atributo nome, do tipo $\mathbf{T}_{cliente}$, é monovalorado atômico;
- O atributo enderecoEntrega, do tipo $\mathbf{T}_{pedido}$, é monovalorado estruturado cujo valor é uma instância do tipo $\mathbf{T}_{endereco}$;
- O atributo cliente_ref, do tipo $\mathbf{T}_{pedido}$, é monovalorado de referência cujo valor é uma referência para uma instância do tipo $\mathbf{T}_{cliente}$;
- O atributo listaItens, no tipo $\mathbf{T}_{pedido,}$ é multivalorado (Nested Table) de tipo $\mathbf{T}_{lista\_item}$ cujo valor é uma coleção de instâncias de $\mathbf{T}_{item}$.

```
CREATE TYPE Tcliente AS OBJECT
( codigo INTEGER, nome VARCHAR2(50) );

CREATE TYPE Titem AS OBJECT
( codigo INTEGER, produto VARCHAR2(30),
  quantidade INTEGER );

CREATE TYPE Tendereco AS OBJECT
( rua VARCHAR2(30), cidade VARCHAR2(15),
  estado VARCHAR2(2), cep VARCHAR2(8) );

CREATE TYPE Tlista_item AS TABLE OF TItem ;
```

```
CREATE TYPE Tpedido AS OBJECT
( codigo INTEGER, data DATE, dataEntrega DATE,
  enderecoEntrega Tendereço, cliente REF Tcliente,
  listaItens Tlista_item);

CREATE TABLE Clientes OF Tcliente
( codigo PRIMARY KEY );

CREATE TABLE Pedidos OF Tpedido
( codigo PRIMARY KEY, cliente REFERENCES Clientes)
NESTED TABLE listaItens STORE AS listaItens_nt_tab;
```

Figura 2.1: Esquema objeto-relacional Pedidos.

O modelo objeto-relacional do *Oracle* 9i suporta dois tipos de tabelas: tabelas de tuplas e tabelas de objetos. As tabelas de tuplas são as tabelas do modelo relacional. Uma tabela de objetos possui associado um tipo e os objetos inseridos na tabela possuem um identificador único (OID), permitindo, assim, que os objetos possam ser referenciados por outros objetos através de atributos de referência. A tabela Clientes no esquema Pedidos da

Figura 2.1 é uma tabela de objetos, cujos objetos são instâncias do tipo $\mathbf{T}_{cliente}$. Estes objetos são referenciados pelas instâncias de $\mathbf{T}_{pedido}$ através do atributo cliente_ref. O escopo de um atributo de referência (tabela ou visão que contém os objetos referenciados) é especificado com as restrições de escopo [33]. Por exemplo, na tabela Pedidos, a restrição referencial cliente_ref REFERENCES Clientes especifica que o escopo do atributo de referência cliente_ref é a tabela Clientes.

Antes de inserir um objeto em uma tabela de objetos devemos criá-lo usando um construtor de objetos. Um construtor de objetos é uma função que cria um objeto de um determinado tipo. Considere, por exemplo, o esquema Pedidos apresentado na Figura 2.1. Um exemplo de construtor de objeto para o tipo $\mathbf{T}_{cliente}$ é $\mathbf{T}_{cliente}$ (1 , "Lineu"). Um objeto do tipo $\mathbf{T}_{lista\_item}$ pode ser criado com o construtor $\mathbf{T}_{lista\_item}$ ($\mathbf{T}_{item}$ (20, 'Coca-Cola', 10), $\mathbf{T}_{item}$ (21, 'Skol', 10)). Observe que $\mathbf{T}_{lista\_item}$ é uma Nested Table que contém duas instâncias do tipo $\mathbf{T}_{item}$.

O modelo objeto-relacional do Oracle 9i permite a criação de tipos pertencentes a uma hierarquia através das cláusulas NOT FINAL e UNDER. Durante o mapeamento de uma hierarquia de tipos, o supertipo é definido adicionando-se a cláusula NOT FINAL à sua declaração do tipo. Essa cláusula permite que subtipos sejam definidos a partir do tipo que está sendo definido. Cada subtipo é então definido utilizando a cláusula UNDER, seguida do seu supertipo imediato na hierarquia.

2.2 Notação Gráfica

Neste trabalho usamos uma notação gráfica baseada em [65] para representar um esquema objeto relacional. A notação adotada estende o diagrama de classes da UML, através do uso de *estereótipos* de forma a permitir modelar, além de tipo de objetos (chamado de classe na UML), também tabelas de objetos, tabelas relacionais e visões de objetos. A Figura 2.2 mostra a representação gráfica do esquema objeto-relacional Pedidos da Figura 2.1. Na notação gráfica, os retângulos com o estereótipo <<Tipo de Objeto>> representam um TAD; os retângulos com o estereótipo <<Tabela de Objeto>> representam

tabelas de objetos; os retângulos com o estereótipo <<Tabela Relacional>> representam tabelas de tuplas; e os retângulos com o estereótipo <<Visão de Objeto>> representam uma tabela virtual de objetos. Os atributos cujos tipos são pré-definidos são escritos dentro dos retângulos. Os demais atributos cujos tipos são definidos pelo usuário são representados por setas rotuladas com o nome do atributo, alem da sua multiplicidade. Para os atributos de referência, adicionamos o estereótipo <<REF>> à seta.

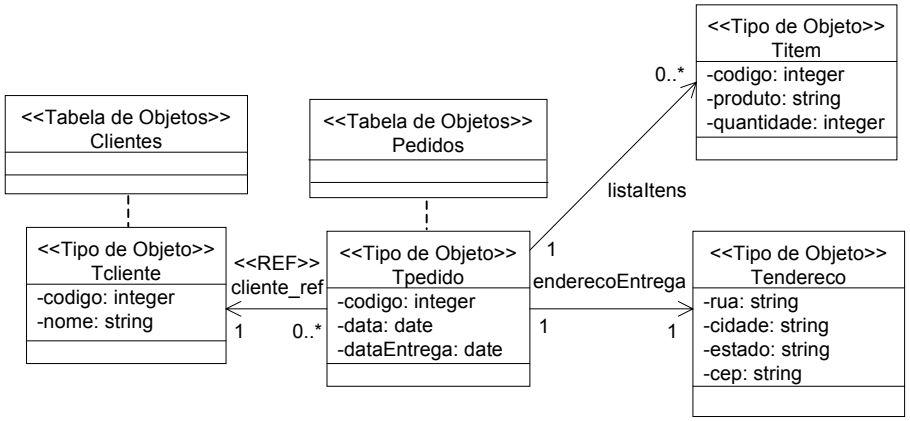


Figura 2.2: Representação gráfica do esquema Pedidos.

2.3 Consulta dos Dados Objeto-relacionais

SQL (*Structured Query Language*) é a linguagem padrão para a definição e manipulação de dados armazenados em banco de dados relacionais. Ela pode ser considerada uma das maiores razões para o sucesso dos bancos de dados relacionais [16]. Em sua mais recente versão (SQL:1999 ou SQL3), adicionou-se novas características à linguagem para permitir o suporte a dados orientados a objetos. Alguns SGBD's, como DB2 e *Oracle*, já dão suporte às características de orientação a objetos da SQL:1999.

Algumas das funcionalidades que caracterizam SQL:1999 como uma linguagem de definição de dados (DDL), orientados a objetos, foram apresentadas na seção anterior (TAD, tabela de objetos, OID, atributo multivalorado, atributo de referência). A seguir,

discutiremos alguns operadores da SQL:1999 como uma linguagem de manipulação de dados (DML) orientados a objeto:

- ***TABLE***

  Esse operador permite desaninhar uma coleção de objetos (*Nested Table* ou *Varray*) dentro de uma cláusula FROM. Dessa forma, conseguimos acessar individualmente cada objeto de uma coleção. Considere, por exemplo, o esquema Pedidos apresentado na Figura 2.2. Para retornar o nome de todos os produtos do pedido de número 1, devemos criar a seguinte consulta SQL:1999:

```
SELECT i.produto
FROM Pedidos p,
        TABLE (p.listaItens) i
WHERE p.codigo = 1
```

- ***REF***

  Esse operador pode ser usado em consultas para obter uma referência para um objeto. Considere, por exemplo, o esquema Pedidos apresentado na Figura 2.2. A consulta a seguir retorna o OID de todos os clientes que fizeram algum pedido:

```
SELECT REF(p)
FROM Pedidos p
```

- ***CAST-MULTISET***

  Esse operador oferece uma forma de sintetizar coleções de objetos (*Nested Table* ou *Varray*) em consultas. Considere, por exemplo, o esquema relacional Pedidos_rel (Figura 2.6) e o tipo coleção $\mathbf{T}_{ped}$ definido pelo comando "CREATE TYPE $\mathbf{T}_{ped}$ AS TABLE OF Integer". Na consulta abaixo, para cada cliente será retornado seu nome e uma coleção de objetos contendo os códigos dos seus pedidos:

```
SELECT c.cnome, CAST ( MULTISET (
                        SELECT p.pcodigo
                        FROM Pedidos_rel p
                        WHERE c.ccodigo = p.pcliente
                ) AS Tped)
FROM Clientes_rel c
```

– ***CURSOR***

O operador **CAST-MULTISET** é fortemente tipado, ou seja, exige que um tipo coleção seja definido para ele. O operador **CURSOR** possui a mesma funcionalidade do operador **MULTISET**, no entanto, não precisa que um tipo seja definido para ele. A consulta abaixo corresponde à consulta anterior, reescrita com o operador **CURSOR**:

```
SELECT c.cnome,CURSOR( SELECT p.pcodigo
                       FROM Pedidos_rel p
                       WHERE c.ccodigo = p.pcliente)
FROM Clientes_rel c
```

2.4 Visões de Objetos

Um banco de dados geralmente é compartilhado por uma variedade de usuários e deve atender a diferentes requisitos desses usuários. Dessa forma, muitas vezes, os bancos de dados tornam-se grandes e complexos, dificultando assim a sua manipulação. Para facilitar a manipulação dos dados, devem ser fornecidas interfaces que apresentem somente as informações relevantes para cada grupo de usuários. Isto é feito através da definição de visões, que representam modelos simplificados do banco de dados, através dos quais os usuários podem expressar consultas e atualizações. As visões, além de protegerem o acesso aos dados, ajudam a alcançar um certo grau de independência lógica, uma vez que é possível alterar o esquema do banco de dados sem alterar uma visão.

Em banco de dados relacionais, as visões relacionais são tabelas virtuais que têm seus dados derivados das tabelas relacionais nas quais estão baseadas, chamadas tabelas base da visão. Assim como uma visão relacional é uma tabela virtual, uma visão de objetos é uma tabela virtual de objetos. Além das características das visões tradicionais, as visões de objetos possuem as seguintes características:

- **Habilidade para navegar usando referências**

  Assim como nas tabelas de objetos, cada objeto construído a partir de uma visão de objetos possui um OID. Dessa forma, os objetos de uma visão de objetos podem ser referenciados por outros objetos. Utilizar referências para representar conexões entre objetos torna as consultas baseadas em caminhos de referência mais legíveis e compactas do que as consultas feitas com junções.

- **Facilidade para evolução de esquemas**

  As visões de objetos fornecem a flexibilidade de ver, de mais de uma maneira, o mesmo dado relacional ou orientado a objetos. Sendo assim, cada aplicação pode ter seu próprio esquema orientado a objetos sem ter quer mudar o esquema do banco de dados.
- **Consistência e compartilhamento de dados relacionais com novas aplicações baseadas em objetos**
- **Integração de aplicações OO com bancos de dados**

  Visões de objeto provêem uma técnica poderosa para se impor visões lógicas, ricamente estruturadas, sobre banco de dados já existentes. Dessa forma possibilitam a coexistência de aplicações relacionais e orientadas a objetos [53][54], o que torna fácil a introdução de aplicações orientadas a objetos para dados relacionais já existentes sem provocar uma mudança drástica de um paradigma para o outro.

Uma visão de objeto possui um tipo de objeto associado, de modo que os objetos da visão são instâncias desse tipo. No *Oracle* 9i, para criar uma visão de objetos, primeiro deve-se criar os tipos da visão. A Figura 2.3 contém a definição dos tipos de uma visão Pedidos_v, cuja representação gráfica é mostrada na Figura 2.4. Depois, então, define-se a visão através de uma consulta SQL:1999 que especifica como os objetos da visão são sintetizados a partir dos dados do banco de dados. A Figura 2.5 contém a consulta SQL:1999 que define a visão Pedido_v baseada no esquema do banco de dados relacional Pedido_rel, apresentado na Figura 2.6. A visão Pedidos_v contém um conjunto de objetos do tipo $\mathbf{T}_{\text{Pedido}}$, os quais são sintetizados a partir das tuplas das tabelas base.

```
CREATE TYPE Tcliente_v AS OBJECT
( codigo INTEGER, nome VARCHAR2(50) );

CREATE TYPE Titem_v AS OBJECT
( codigo INTEGER, produto VARCHAR2(30),
    quantidade INTEGER );

CREATE TYPE Tendereco_v AS OBJECT
( rua VARCHAR2(30), cidade VARCHAR2(15),
  estado VARCHAR2(2), cep VARCHAR2(8) );

CREATE TYPE Tlista_item_v AS TABLE OF Titem_v ;

CREATE TYPE Tpedido_v AS OBJECT
( codigo INTEGER, data DATE, dataEntrega DATE,
  enderecoEntrega Tendereco_v, cliente_ref REF Tcliente_v,
  listaItens Tlista_item_v);
```

Figura 2.3: Definição dos tipos da visão Pedidos_v.

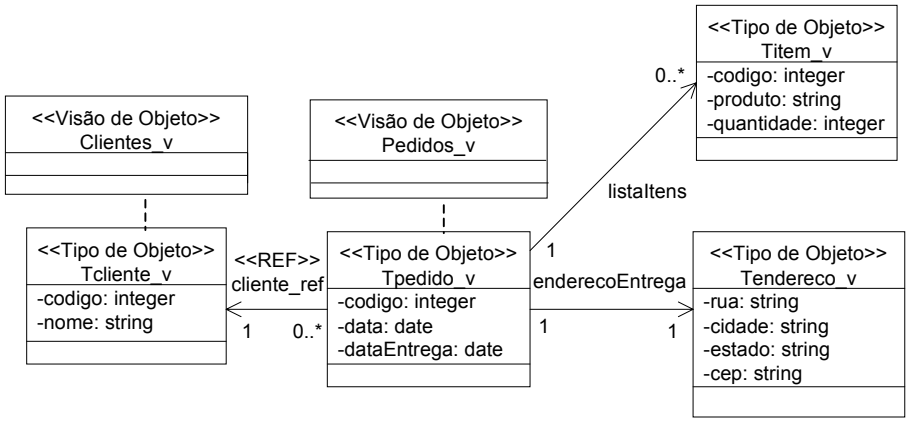


Figura 2.4: Esquema da visão de objetos Pedidos_v.

```
CREATE VIEW Pedidos_v OF Tpedido
WITH OBJECT IDENTIFIER (codigo) AS
   SELECT

Tpedido(
        tpedidos_rel•pcodigo,
        tpedidos_rel•pdata,
        tpedidos_rel•pdataEntrega,
        Tendereco ( tpedidos_rel•prua, tpedidos_rel•pcidade, tpedidos_rel•pestado, tpedidos_rel•pcep ),
        (SELECT REF (tcliente_v)
         FROM Clientes_v tcliente_v, Clientes_rel tclientes_rel
         WHERE tpedidos_rel•pcliente = tclientes_rel•ccodigo
                tclientes_rel•ccodigo = tcliente•codigo),
        CAST(MULTISET( SELECT

                              Titem_v(
                                    titens_rel•icodigo,
                                    (SELECT tprodutos_rel•pnome
                                     FROM Produtos_rel tprodutos_rel
                                     WHERE titens_rel•iproduto= tprodutos_rel•pcodigo),
                                    titens_rel•iquantidade)
                              FROM Itens_rel titens_rel
        ) AS Tlista_item_v )

FROM Pedidos_rel tpedidos_rel
```

Figura 2.5: Definição da visão Pedidos_v.

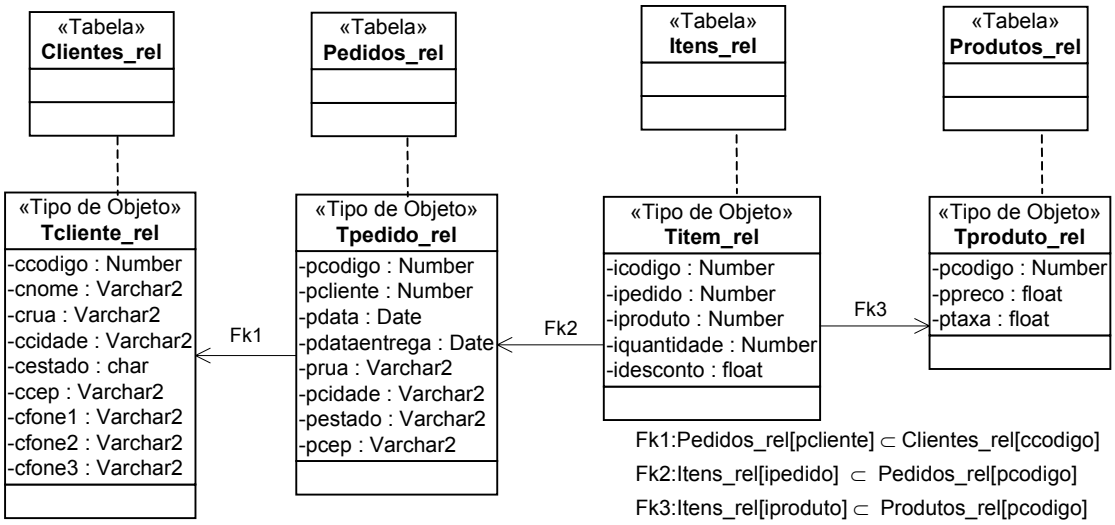


Figura 2.6: Esquema do banco de dados Pedidos_rel.

Consultas SQL:1999, definidas sobre uma visão de objetos, são traduzidas em consultas sobre o banco de dados. Os resultados dessas consultas são, então, integrados para gerar o resultado da consulta inicial [18]. No caso em que atualizações sobre uma visão de objetos são permitidas, deve-se definir tradutores (*instead of triggers*) que especifiquem como atualizações sobre a visão de objetos são traduzidas em atualizações especificadas sobre o banco de dados. O código do *instead of trigger* é executado quando inserções, atualizações ou remoções são solicitadas na visão ou em algum atributo multivalorado da visão. A Figura 2.7 mostra, a título de ilustração, o *instead of trigger* "Adiciona_Cliente", o qual traduz a adição de um pedido na visão Clientes_v (Figura 2.4) em uma adição na tabela relacional Clientes_rel (Figura 2.6).

```
CREATE TRIGGER Adiciona_Cliente
INSTEAD OF TRIGGER INSERT ON Clientes_v
FOR EACH ROW
   BEGIN
      INSERT INTO CLIENTES_REL
      (ccodigo, cnome)
       VALUES (:new.codigo, :new.nome)
  END
```

Figura 2.7: *Instead of trigger* Adiciona_Cliente.

2.5 Assertivas de Correspondência no Modelo Objeto-Relacional

Nesta seção, apresentamos a definição formal dos vários tipos de assertivas de correspondência que são utilizadas para especificar formalmente as correspondências entre esquemas objeto-relacionais. O formalismo proposto permite especificar várias formas de correspondência, inclusive casos onde elementos semelhantes são representados de formas diferentes. As Assertivas de Correspondência (AC) são classificadas em: (i) AC de Extensão, (ii) AC de Caminhos e (iii) AC de Objetos. A seguir, apresentamos algumas definições necessárias para a definição formal dos tipos de assertivas citados acima.

2.5.1 Terminologias

Definição 4.1: Usamos o conceito de **ligação** para representar os relacionamentos entre tipos. Considere $\mathbf{T_1}$ e $\mathbf{T_2}$ tipos de um esquema. Existe uma ligação de $\mathbf{T_1}$ para $\mathbf{T_2}$ nas situações descritas a seguir:

(i) (**Ligação de atributo de valor**) $\mathbf{T_1}$ contém um atributo **a** cujo tipo é $\mathbf{T_2}$ ou coleção de objetos do tipo $\mathbf{T_2}$. Assim, **a**: $\mathbf{T_1 \rightarrow T_2}$ é uma ligação de $\mathbf{T_1}$ para $\mathbf{T_2}$. Dada uma instância $\mathbf{t_1}$ de $\mathbf{T_1}$, a expressão $\mathbf{t_1 \bullet a}$ retorna o valor do atributo **a** (uma instância ou coleção de instâncias de $\mathbf{T_2}$).

(ii) (**Ligação de atributo de referência**) $\mathbf{T_1}$ contém um atributo **a** cujo tipo é uma referência ou uma coleção de referências para objetos do tipo $\mathbf{T_2}$. Assim, **a**: $\mathbf{T_1 \rightarrow T_2}$ é uma ligação de $\mathbf{T_1}$ para $\mathbf{T_2}$. Dada uma instância $\mathbf{t_1}$ de $\mathbf{T_1}$, a expressão $\mathbf{t_1 \bullet a}$ retorna as instâncias de $\mathbf{T_2}$ referenciadas por $\mathbf{t_1}$ através do atributo **a**.

(iii)(**Ligação de chave estrangeira**) Existe uma chave estrangeira $\mathbf{fk = R_1[a_1,...,a_n] \subseteq R_2[b_1,...,b_n]}$, onde $\mathbf{T_1}$ e $\mathbf{T_2}$ são os tipos[1] das tuplas de $\mathbf{R_1}$ e $\mathbf{R_2}$, respectivamente. Assim, **fk**: $\mathbf{T_1 \rightarrow T_2}$ é uma ligação de $\mathbf{T_1}$ para $\mathbf{T_2}$. Dada uma tupla $\mathbf{t_1}$ de $\mathbf{R_1}$, a expressão $\mathbf{t_1 \bullet fk}$ retorna a tupla $\mathbf{t_2}$ de $\mathbf{R_2}$ tal que $\mathbf{t_1 \bullet a_i = t_2 \bullet b_i}$, para $1 \leq i \leq n$.

[1] Para tratarmos de forma uniforme tabelas de objetos e tabelas de tuplas, assumimos que dada uma tabela de tuplas **R**, $\mathbf{T_R}$ é o tipo das tuplas de **R**. Este tipo é na verdade, definido implicitamente na própria criação da tabela **R**.

(iv)(**Inversa de ligação**) T_2 tem uma ligação l: $T_2 \rightarrow T_1$ tal que l é uma ligação de atributo de referência ou de chave estrangeira. Então, a inversa da ligação l, dada por l^{-1}: $T_1 \rightarrow T_2$ é uma ligação de T_1 para T_2. Este tipo de ligação é chamado de **ligação virtual**. Os demais tipos de ligações são chamados de **ligação direta**.

Definição 4.2: Um objeto pode estar relacionado com outro objeto através da composição de duas ou mais ligações. Considere as ligações, l_1: $T_1 \rightarrow T_2$, l_2: $T_2 \rightarrow T_3$,..., l_{n-1}: $T_{n-1} \rightarrow T_n$, onde T_1, T_2,..., T_n são tipos de um esquema objeto-relacional. Então, $\varphi = l_1 \bullet l_2 \bullet ... \bullet l_{n-1}$ é um caminho de T_1. Isso significa que as instâncias de T_1 estão relacionadas com as instâncias de T_n através do caminho φ. Dada uma instância t_1 de T_1, a expressão $t_1 \bullet \varphi$ retorna objetos t_n de T_n tais que existem as instâncias t_2, t_3,..., t_{n-1} onde t_i está relacionada com t_{i+1} através da ligação l_i, para $1 \leq i < n$. Assim, o tipo do caminho φ é T_n ($T_\varphi = T_n$). Se as ligações l_1, l_2,..., l_{n-1} são monovaloradas, então φ é um caminho monovalorado, caso contrário φ é um caminho multivalorado. Se l_{n-1} é uma ligação de referência então φ é um caminho de referência, caso contrário φ é um caminho de valor.

2.5.2 Assertivas de Correspondência de Extensão

A extensão de uma tabela (ou visão) é o conjunto de objetos que são membros da tabela (ou visão) em um determinado instante. As Assertivas de Correspondência de Extensão (ACE) especificam a correspondência existente entre a extensão da visão com a extensão de sua tabela base. No resto desta seção, considera-se **V** uma visão e **R** uma tabela. Existem dois tipos de ACE's, definidas a seguir.

Definição 4.3: A ACE $[\mathbf{V}] \equiv [\mathbf{R}]$ especifica que $\mathbf{t_v} \in \mathbf{V}$ sss $\exists\ \mathbf{t_R} \in \mathbf{R}$ tal que $\mathbf{t_v}$ e $\mathbf{t_R}$ são "semanticamente equivalentes" ($\mathbf{t_v} \equiv \mathbf{t_R}$), i.e. correspondem ao mesmo objeto no mundo real.

Definição 4.4: A ACE $[\mathbf{V}] \equiv [\mathbf{R}[P]]$, onde P é um predicado condicional, especifica que $\mathbf{t_v} \in \mathbf{V}$ sss $\exists\ \mathbf{t_R} \in \mathbf{R}$ tal que $\mathbf{t_R}$ satisfaz a condição P e $\mathbf{t_v} \equiv \mathbf{t_R}$.

2.5.3 Assertivas de Correspondência de Caminho

As ACC's especificam os relacionamentos entre atributos do tipo da visão com atributos/caminhos do tipo da tabela base, denominado de **tipo base**. No restante desta seção, considere T_v o tipo da visão **V** e T_b seu tipo base.

Definição 4.5: Seja **c** um atributo de T_v e φ um caminho de T_b, onde a cardinalidade de **c** é igual a cardinalidade de φ. A ACC $[T_v \bullet c] \equiv [T_b \bullet \varphi]$ especifica que para quaisquer instâncias t_v de T_v e t_b de T_b, se $t_v \equiv t_b$ então $t_v \bullet c \equiv t_b \bullet \varphi$. No caso em que **c** é monovalorado, dados $t'_v = t_v \bullet \varphi_v$ e $t'_b = t_b \bullet \varphi$ então $t'_v \equiv t'_b$. No caso em que **c** é multivalorado, temos que t'_v pertence a $t_v \bullet \varphi_v$ sss existe t'_b em $t_b \bullet \varphi$ tal que $t'_v \equiv t'_b$.

Definição 4.6. Suponha $b_1, b_2, ..., b_n$ atributos de valor atômico de T_b e **a** um atributo de T_v, cujo tipo T_a contém atributos atômicos $a_1, a_2, ..., a_n$. A ACC $[T_v \bullet a, \{a_1, a_2, ..., a_n\}] \equiv [T_b, \{b_1, b_2, ..., b_n\}]$ especifica que dada uma instância t_v de T_v e t_b de T_b, se $t_v \equiv t_b$ então $t_v \bullet a \bullet a_i = t_b \bullet b_i$, para $1 \leq i \leq n$.

Definição 4.7. Seja **a** um atributo multivalorado de valor atômico de T_v e φ um caminho monovalorado de T_b, de tipo T_φ. Sejam $c_1, c_2, ..., c_n$ atributos monovalorados de valores atômicos de T_φ. A ACC $[T_v \bullet a] \equiv [T_b \bullet \varphi, \{c_1, c_2, ..., c_n\}]$ especifica que para quaisquer instâncias t_v de T_v e t_b de T_b, se $t_v \equiv t_b$ então t'_v pertence a $t_v \bullet a$ sss existe um objeto t'_b em $\{t_b \bullet \varphi \bullet c_1, t_b \bullet \varphi \bullet c_2, ..., t_b \bullet \varphi \bullet c_n\}$, tal que $t'_v = t'_b$.

2.5.4 Assertivas de Correspondência de Objeto

As assertivas de correspondências de objetos (ACO) especificam sob que condições dois objetos, os quais são instâncias de tipos semanticamente relacionados, representam o mesmo objeto do mundo real, ou seja, são semanticamente equivalentes.

Definição 4.8. Considere os tipos $\mathbf{T_v}$ e $\mathbf{T_b}$ os quais são semanticamente relacionados. Sejam $\mathbf{v}_1,...,\mathbf{v}_n$ atributos de $\mathbf{T_v}$ e $\mathbf{b}_1,...,\mathbf{b}_n$ atributos de $\mathbf{T_b}$. A ACO $[\mathbf{T_v}, \{ \mathbf{v}_1,...,\mathbf{v}_n \}] \equiv [\mathbf{T_b}, \{ \mathbf{b}_1,...,\mathbf{b}_n \}]$ especifica que para qualquer instância $\mathbf{t_v}$ de $\mathbf{T_v}$ e $\mathbf{t_b}$ de $\mathbf{T_b}$, se $\mathbf{t_v \bullet v_i = t_b \bullet b_i}$, $1 \leq i \leq n$, então $\mathbf{t_v \equiv t_b}$.

2.5.5 Usando Assertivas de Correspondência para Especificar Visões de Objeto

Neste trabalho, uma visão de objeto **V** sobre o esquema S é definida por uma 4-tupla $\mathbf{V} = < \mathbf{T_v}, \psi_V, C_V, A_V >$ onde $\mathbf{T_v}$ é o tipo dos objetos de **V**, ψ_V é uma ACE, C_V é um conjunto de ACC e A_V é um conjunto de ACO. Isso define um mapeamento funcional, denotado DEF_V, de estados σ_S de S em estados da visão, como definido a seguir:

Caso 1: Para ψ_v da forma $[\mathbf{V}] \equiv [\mathbf{R}[P]]$, temos:

$$DEF_V(\sigma_S) = \{ ConstrutorT_v\_T_R(\mathbf{t}) \mid \mathbf{t} \in \sigma_S(R) \text{ e } P(\mathbf{t}) = true \},$$

onde $ConstrutorT_v\_T_R(\mathbf{t})$ cria uma instância **v** de $\mathbf{T_V}$ tal que $\mathbf{v} \equiv \mathbf{t}$. Note que se $\mathbf{v} \equiv \mathbf{t}$ então **v** deve satisfazer a todas as ACC's de $\mathbf{T_v \& T_b}$. Assim, os valores dos atributos do objeto **v** criado são definidos de acordo com as assertivas de correspondência de caminho de $\mathbf{T_v}$ & $\mathbf{T_R}$. Em [42] é proposto um algoritmo que gera automaticamente o construtor de objetos da visão, a partir das ACC's da visão.

Caso 2: Para ψ_V da forma $[\mathbf{V}] \equiv [\mathbf{R}]$, temos:

$$DEF_V(\sigma_S) = \{ ConstrutorT_v\_T_R(\mathbf{t}) \mid t \in \sigma_S(R) \}.$$

Exemplo 2.1

No resto desta seção, considere a visão **Pedidos_v** cujos objetos são do tipo $\mathbf{T_{pedido}}$ (vide Figura 2.4) e o esquema do banco de dados **Pedidos_rel,** apresentado na Figura 2.6. Considere que a ACE de **Pedidos_v** é dada por ψ_1:$[\mathbf{Pedidos\_v}] \equiv [\mathbf{Pedidos\_rel}]$. ψ_1 especifica

que as extensões da visão **Pedidos_v** e da tabela **Pedidos_rel** denotam o mesmo conjunto de objetos do mundo real. As ACC's de **Pedidos_v** são mostradas na Figura 2.8.

ACC's de $\mathbf{T}_{pedido\_v}$ & $\mathbf{T}_{pedidos\_rel}$
ψ_2: $[\mathbf{T}_{pedido\_v}\bullet codigo] \equiv [\mathbf{T}_{pedidos\_rel}\bullet pcodigo]$
ψ_3: $[\mathbf{T}_{pedido\_v}\bullet data] \equiv [\mathbf{T}_{pedidos\_rel}\bullet pdata]$
ψ_4: $[\mathbf{T}_{pedido\_v}\bullet dataEntrega] \equiv [\mathbf{T}_{pedidos\_rel}\bullet pdataEntrega]$
ψ_5: $[\mathbf{T}_{pedido\_v}\bullet enderecoEntrega], \{rua, cidade, estado, cep\}] \equiv [\mathbf{T}_{pedidos\_rel}, \{prua, pcidade, pestado, pcep\}]$
ψ_6: $[\mathbf{T}_{pedido\_v}\bullet cliente\_ref] \equiv [\mathbf{T}_{pedidos\_rel}\bullet fk_1]$
ψ_7: $[\mathbf{T}_{pedido\_v}\bullet listaItens] \equiv [\mathbf{T}_{pedidos\_rel}\bullet fk_2^{-1}]$

ACC's de $\mathbf{T}_{cliente\_v}$ & $\mathbf{T}_{clientes\_rel}$
ψ_9: $[\mathbf{T}_{cliente\_v}\bullet codigo] \equiv [\mathbf{T}_{clientes\_rel}\bullet ccodigo]$
ψ_{10}: $[\mathbf{T}_{cliente\_v}\bullet nome] \equiv [\mathbf{T}_{clientes\_rel}\bullet cnome]$
ψ_{11}: $[\mathbf{T}_{cliente\_v}\bullet telefones] \equiv [\mathbf{T}_{clientes\_rel}, \{cfone1, cfone2, cfone3\}]$

ACC's de $\mathbf{T}_{item\_v}$ & $\mathbf{T}_{itens\_rel}$
ψ_{13}: $[\mathbf{T}_{item\_v}\bullet codigo] \equiv [\mathbf{T}_{itens\_rel}\bullet icodigo]$
ψ_{14}: $[\mathbf{T}_{item\_v}\bullet produto] \equiv [\mathbf{T}_{itens\_rel}\bullet fk_3\bullet pnome]$
ψ_{15}: $[\mathbf{T}_{item\_v}\bullet quantidade] \equiv [\mathbf{T}_{itens\_rel}\bullet iquantidade]$

Figura 2.8: ACC's de Pedidos_v

O processo de geração das ACC's é *top down* e recursivo. Primeiro, definimos as ACC's dos atributos de $\mathbf{T}_{pedido\_v}$ com atributos/caminhos da tabela pivô **Pedidos_rel**. No caso de atributos de referência ou de tipos complexo deve-se então recursivamente definir as ACC's dos seus atributos com o seu tipo base.

Das ACC's de $\mathbf{T}_{pedido\_v}$&$\mathbf{T}_{pedidos\_rel}$ temos que dada uma instância $\mathbf{t}_v$ de $\mathbf{T}_{pedido\_v}$, e uma instância $\mathbf{t}_b$ de $\mathbf{T}_{pedidos\_rel}$ tal que $\mathbf{t}_v \equiv \mathbf{t}_b$ então:

(i) $\mathbf{t}_v\bullet codigo = \mathbf{t}_b\bullet pcodigo$ (de ψ_2)

(ii) $\mathbf{t}_v\bullet data = \mathbf{t}_b\bullet pdata$ (de ψ_3)

(iii) $\mathbf{t}_v\bullet$ dataEntrega = $\mathbf{t}_b\bullet pdataEntrega$ (de ψ_4)

(iv) $\mathbf{t}_v\bullet enderecoEntrega = \mathbf{T}_{endereco\_v}(\mathbf{t}_b\bullet prua, \mathbf{t}_b\bullet pcidade, \mathbf{t}_b\bullet pestado, \mathbf{t}_b\bullet pcep)$ (de ψ_5)

(v) $\mathbf{t}_v\bullet$ cliente_ref $\equiv \mathbf{t}_b\bullet \mathbf{Fk}_1$ (de ψ_6). Como o tipo de cliente_ref é uma referência para $\mathbf{T}_{cliente\_v}$, que é um tipo estruturado, deve-se definir as ACC's de $\mathbf{T}_{cliente\_v}$ & $\mathbf{T}_{clientes\_rel}$.

(vi) $\mathbf{t'} \in \mathbf{t_v}\bullet$listaItens sss existe $\mathbf{t} \in \mathbf{t_b}\bullet\mathbf{Fk_2}^{-1}$ e $\mathbf{t} \equiv \mathbf{t'}$ (de ψ_7). Como listaItens é uma coleção de objetos de tipo $\mathbf{T}_{item\_v}$, que é um tipo estruturado, deve-se definir as ACC's de $\mathbf{T}_{item\_v}$ & $\mathbf{T}_{itens\_rel}$.

Das ACC's de $\mathbf{T}_{cliente\_v}$ & $\mathbf{T}_{clientes\_rel}$ temos que dada uma instância $\mathbf{t_v}$ de $\mathbf{T}_{cliente\_v}$, e uma instância $\mathbf{t_b}$ de $\mathbf{T}_{clientes\_rel}$ tal que $\mathbf{t}_v \equiv \mathbf{t_b}$ então:

(i) $\mathbf{t_v}\bullet$codigo = $\mathbf{t_b}\bullet$ccodigo (de ψ_9)

(ii) $\mathbf{t_v}\bullet$nome = $\mathbf{t_b}\bullet$cnome (de ψ_{10})

(iii) $\mathbf{t_v}\bullet$ telefones = $\mathbf{T}_{lista_fone}(\mathbf{t_b}\bullet$cfone1, $\mathbf{t_b}\bullet$cfone2, $\mathbf{t_b}\bullet$cfone2) (de ψ_{11})

Das ACC's de $\mathbf{T}_{item\_v}$ & $\mathbf{T}_{itens\_rel}$ temos que dada uma instância $\mathbf{t_v}$ de $\mathbf{T}_{item\_v}$, e uma instância $\mathbf{t_b}$ de $\mathbf{T}_{itens\_rel}$ tal que $\mathbf{t_v} \equiv \mathbf{t_b}$ então:

(i) $\mathbf{t_v}\bullet$codigo = $\mathbf{t_b}\bullet$icodigo (de ψ_{13})

(ii) $\mathbf{t_v}\bullet$produto = $\mathbf{t_b}\bullet\mathbf{Fk}_3\bullet$pnome (de ψ_{14})

(iii) $\mathbf{t_v}\bullet$ quantidade = $\mathbf{t_b}\bullet$iquantidade (de ψ_{15})

As assertivas de correspondência de objeto de **Pedidos_v** são mostradas na Figura 2.9. As ACO's são inferidas a partir das ACC's e das chaves primárias das tabelas do banco de dados. Por exemplo, da ACC ψ_2 e da chave primária da tabela **Pedidos_rel**, inferimos a ACO ψ_8: $[\mathbf{T}_{pedido\_v}, \{codigo\}] \equiv [\mathbf{T}_{pedidos\_rel}, \{pcodigo\}]$ que especifica que para quaisquer instâncias $\mathbf{t_v}$ de $\mathbf{T}_{pedido\_v}$ e $\mathbf{t_b}$ de $\mathbf{T}_{pedidos\_rel}$, se $\mathbf{t_v}\bullet$codigo = $\mathbf{t_b}\bullet$pcodigo, então $\mathbf{t_v} \equiv \mathbf{t_b}$.

ACO de $\mathbf{T}_{pedido\_v}$ & $\mathbf{T}_{pedidos\_rel}$
ψ_8: $[\mathbf{T}_{\mathbf{pedido\_v}}, \{codigo\}] \equiv [\mathbf{T}_{pedidos\_rel}, \{pcodigo\}]$

ACO de $\mathbf{T}_{cliente\_v}$ & $\mathbf{T}_{clientes\_rel}$
ψ_{12}:$[\mathbf{T}_{cliente\_v}, \{codigo\}] \equiv [\mathbf{T}_{clientes\_rel}, \{ccodigo\}]$

ACO de $\mathbf{T}_{Item\_v}$ & $\mathbf{T}_{itens\_rel}$
ψ_{16}: $[\mathbf{T}_{item\_v}, \{codigo\}] \equiv [\mathbf{T}_{itens\_rel}, \{icodigo\}]$

Figura 2.9: ACO's de Pedidos_v

Como é mostrado em [42], a definição SQL:1999 de uma visão de objetos pode ser gerada automaticamente a partir das AC's da visão. A Figura 2.10 mostra a definição SQL:1999 da visão que realiza o mapeamento definido pelas assertivas.

```
CREATE VIEW Pedidos_v OF T_pedido
WITH OBJECT IDENTIFIER (codigo) AS
   SELECT

1. T_pedido_v(
2.        t_pedidos_rel•pcodigo,                                              (de ψ3 )
3.        t_pedidos_rel•pdata,                                                (de ψ4 )
4.        t_pedidos_rel•pdataEntrega,                                         (de ψ5 )
5.        T_endereco ( t_pedidos_rel•prua, t_pedidos_rel•pcidade, t_pedidos_rel•pestado, t_pedidos_rel•pcep ),   (de ψ6 )
6.        (SELECT REF (t_cliente_v)                                           (de ψ7 )
          FROM Clientes_v t_cliente_v, Clientes_rel t_clientes_rel
          WHERE t_pedidos_rel•pcliente = t_clientes_rel•ccodigo
                t_clientes_rel•ccodigo = t_cliente•codigo),                   (da ACO ψ9)
7.        CAST(MULTISET( SELECT                                               (de ψ8 )

                         T_item_v(
8.                            t_itens_rel•icodigo,                           (de ψ14)
9.                            (SELECT t_produtos_rel•pnome                   (de ψ15)
                               FROM Produtos_rel t_produtos_rel
                               WHERE t_itens_rel•iproduto= t_produtos_rel•pcodigo),
10.                           t_itens_rel•iquantidade)                       (de ψ16)
                         FROM Itens_rel  t_itens_rel
11.       ) AS T_lista_item_v)

FROM Pedidos_rel  t_pedidos_rel
```

Figura 2.10 : Definição da visão de objeto Pedidos_v

Capítulo 3
Publicação de Dados no Formato XML

Nos últimos anos, a Web tem se tornado o maior ambiente capaz de prover acesso a fontes de dados heterogêneas. Nesse contexto, a linguagem XML (*Extensible Markup Language*)[56] vêm se tornando padrão para integração, publicação e troca de dados na Web. Essa crescente popularidade da linguagem se deve à sua habilidade na representação de dados estruturados e semi-estruturados, por ser uma linguagem autodescritível, extensível e bastante flexível, que lhe permite modelar os requisitos exigidos por aplicações complexas. Além disso, a linguagem XML permite uma separação entre os dados que compõem a página e a sua apresentação.

Esse capítulo está organizado como se segue. Na Seção 3.1, discutimos as motivações do uso de XML, apresentando suas principais características. Na Seção 3.2, mostramos como definir esquemas XML através da linguagem XML *Schema*. Na Seção 3.3, abordamos como manipular os documentos XML usando a linguagem *XQuery*. Na Seção 3.4, mostramos como podemos transformar documentos XML usando XSLT. Na Seção 3.5, discutimos como publicar dados armazenados em base de dados relacionais ou objeto-relacionais no formato XML.

3.1 XML

XML (*eXtensible Markup Language*) [56] é uma linguagem de marcação desenvolvida pelo W3C (*World Wide Web Consortium*) [55] para descrever informações. Assim como HTML [57], XML tem origem na SGML (*Standard Generalized Markup Language*), que é um padrão internacional para definição de formatos de representação de texto em meio eletrônico. Entretanto, ao contrário de HTML, que foi projetada para

descrever a apresentação dos dados, XML foi projetada para descrever o conteúdo dos dados. A seguir, destacamos alguns aspectos que tornam a linguagem XML mais poderosa que HTML:

- **Linguagem extensível**. XML permite que os usuários definam novos marcadores, de acordo com o domínio que está sendo modelado. Os marcadores servem para descrever o conteúdo de um documento;
- **Independência de conteúdo e apresentação**. XML aborda apenas o conteúdo de um documento. A apresentação pode ser tratada por linguagens específicas, tais como: *Cascading Style Sheets* (CSS) e *eXtensible Stylesheet Language* (XSL);
- **Linguagem flexível**. XML permite a apresentação de um mesmo conteúdo em diversos formatos;
- **Validação**. Um documento XML pode ser associado a um esquema que define a estrutura do documento. Assim, aplicações podem validar seus dados de acordo com um esquema.

XML também pode ser vista como uma metalinguagem, pois, por ser extensível, permite a criação de outras linguagens de marcadores. Estas linguagens podem ser definidas para domínios específicos (matemática, química, comércio eletrônico, entre outros). Assim, os marcadores podem capturar mais semântica sobre os dados que estão sendo modelados, o que não ocorre em um documento HTML, pois seus marcadores têm função apenas de formatar o texto para apresentação.

Um documento XML consiste em vários elementos. Um elemento é descrito por um marcador inicial (i.e., <nome_do_elemento>) e um final (i.e., </nome_do_elemento>) que delimitam seu conteúdo. Cada elemento pode conter texto e elementos aninhados (sub-elementos), como pode ter conteúdo misto ou ser vazio. Quando o elemento é vazio, este deve ser representado de acordo com um dos seguintes formatos: <nome do elemento></nome do elemento> ou <nome do elemento/>. Considere, por exemplo, o documento XML mostrado na figura a seguir:

```
<pessoa>
        <nome>Tâmara</nome>
        <e-mail>tamara@lia.ufc.br</e-mail>
</pessoa>
```

Figura 3.1: Exemplo de Documento XML.

Os marcadores <pessoa> e </pessoa> descrevem a estrutura do elemento pessoa. Esse elemento é composto por dois sub-elementos: nome e e-mail. Como podemos observar, os marcadores de um documento são balanceados, ou seja, são fechadas na ordem inversa da qual foram abertos.

Um elemento também pode possuir um ou mais atributos. Estes são especificados no marcador inicial do elemento ou, no caso do elemento vazio, no marcador que define o elemento. O valor de um atributo é considerado texto e deve aparecer entre aspas. Por exemplo, no documento XML mostrado abaixo, a propriedade CPF é um atributo do elemento pessoa:

```
<pessoa cpf="8523692205">
    <nome>Tâmara </nome>
    <e-mail>tamara@lia.ufc.br </e-mail>
</pessoa>
```

Figura 3.2: Documento XML com atributo.

Documentos XML podem ser classificados como bem-formados. Para ser bem-formado, um documento deve obedecer às regras sintáticas definidas na especificação da linguagem XML [56], tais como: (1) conter um ou mais elementos; (2) conter um elemento que contenha todos os outros, chamado de elemento raiz e (3) todos os outros elementos devem estar aninhados, sendo delimitados apropriadamente por seus marcadores. Os documentos XML apresentados anteriormente (Figura 3.1 e Figura 3.2) são documentos XML bem-formados. No entanto, o documento XML apresentado na Figura 3.3 é um exemplo de documento XML mal-formado, pois o primeiro elemento-nome não possui um marcador final adequado:

```
<pessoa>
    <nome>Tâmara </nomes>
    <nome>Vânia </nome>
</pessoa>
```

Figura 3.3: Documento XML mal-formado.

Documentos XML também podem ser classificados como válidos. Para ser válido um documento precisa ser bem-formado e deve estar de acordo com a gramática que define sua estrutura. Esta gramática é definida através de um esquema XML, que descreve a

estrutura de um documento, determinando os elementos que podem participar deste documento e também os que podem estar associados a esses elementos. A Seção 3.2 mostra como definir esquemas para documentos XML.

A verificação sintática de um documento XML, que identifica se este é ou não bem-formado, é realizada por um *parser*. Além de detectar se um documento é bem-formado, os *parsers* de validação são capazes de verificar se o mesmo está de acordo com o seu esquema XML associado, ou seja, se o documento é válido ou não.

3.2 Definindo Esquemas para Documentos XML

Um esquema descreve a estrutura lógica de uma fonte de informação, incluindo os tipos dos dados, os relacionamentos e as restrições que envolvem os dados. Um esquema XML é usado para descrever os elementos com seu conteúdo, a lista de atributos de um determinado elemento e as restrições sobre os elementos/atributos. Além disso, um esquema é útil para validar o conteúdo de um documento, ou seja, para determinar se um documento está de acordo com a gramática definida pelo esquema. E essa gramática pode ser reutilizada para validar e definir a estrutura de outros documentos.

A funcionalidade de validar um documento XML é muito importante no contexto de aplicações *web* que trocam informações entre diversas fontes, pois com a definição de esquema é possível verificar se um determinado documento está de acordo com a estrutura esperada, facilitando o processamento de dados pelas aplicações.

A primeira linguagem proposta para definição de esquema XML foi a DTD (*Document Type Definition*) [56]. Entretanto, DTD possui algumas limitações, como pode ser observado em [41]. Para sobrepor essas limitações, outras linguagens foram propostas, entre as quais destacamos: XML *Schema* [61] e RDF [59]. Essas linguagens são mais ricas em semântica e oferecem recursos adicionais para definição de esquemas. Uma análise comparativa dessas e de outras linguagens de esquema XML é apresentada em [27]. A seguir, apresentamos a linguagem XML Schema usada para definir os esquemas XML da aplicação e das VCN's.

3.2.1 XML *Schema*

A linguagem XML *Schema* introduz novos construtores que a tornam mais expressiva que DTD e permite não só especificar a sintaxe de um documento XML, mas também: especificar o tipo de dados do conteúdo de cada elemento; herdar sintaxe de outros esquemas; criar tipos de dados simples e complexos; especificar o número mínimo e máximo de vezes que um elemento pode ocorrer; entre outros. Com isso, XML *Schema* pode ser utilizada em uma maior variedade de aplicações, além de ter a vantagem de ser escrita em sintaxe XML, não sendo necessário, portanto, aprender uma nova sintaxe. Essa facilidade nos permite utilizar qualquer ferramenta que trabalha com documentos XML para trabalhar com XML *Schema* [15].

Na Figura 3.5, apresentamos um exemplo de XML *Schema*. Como pode ser observado, trata-se de um documento XML e o elemento raiz desse documento *Schema*. Em XML *Schema*, todas as declarações de elementos, atributos e tipos devem ser inseridas entre os marcadores do elemento *Schema*.

```
<bib>
 <livro autoresref1="A">
  <ano> 1999 </ano>
  <isbn> 3826561422 </isbn>
  <titulo> Transaction Management in Multidatabase Systems
</titulo>
  <editora> Shaker-Verlag </editora>
 </livro>
 <artigo autoresref="A1 A2">
  <ano> 2000 </ano>
  <cdu> 681.31:061.68 </cdu>
  <titulo> Temporal Serialization Graph Testing </titulo>
  <local_ publicacao> XV SBBD </local_ publicacao>
 </artigo>
 <autor id_autor1="A" instituicaoref="I">
  <nome> Ângelo Brayner </nome>
  <e-mail> brayner@unifor.br </e-mail>
 </autor>
 <autor id_autor="A2" instituicaoref="I1 I2">
  <nome> José Maria Monteiro </nome>
  <e-mail> zemaria@lia.ufc.br </e-mail>
 </autor>
<instituicao id_instituicao="I">
  <nome> Unifor </nome>
  <endereco>
    <cidade> Fortaleza </cidade>
    <estado> Ceará </estado>
    <pai> Brasil </pais>
  </endereco>
 </instituicao>
 <instituicao id_instituicao="I2">
  <nome> UFC </nome>
  <telefone> 288-9845 </telefone>
  <endereco>
    <cidade> Fortaleza </cidade>
    <estado> Ceará </estado>
    <pais> Brasil </pais>
  </endereco>
 </instituicao
</bib>
```

Figura 3.4: bib.xml.

A seguir, descrevemos os principais componentes necessários para a criação de *XML Schemas*. Para exemplificar o uso destes componentes, utilizamos o esquema da Figura 3.5, que representa a estrutura do documento *bib.xml* apresentado na Figura 3.4.

```
1 <Schema>
2  <element name=”bib”>
3   <complexType>
4    <sequence>
5      <element name=”livro” type=”Tlivro” minOccurs=”0” maxOccurs=”unbounded”/>
6    <element name=”artigo” type=”Tartigo” minOccurs=”0” maxOccurs=”unbounded”/>
7     <element name=”autor” type=”Tautor” minOccurs=”0” maxOccurs=”unbounded”/>
8     <element name=”instituicao” type=”Tinstituicao” minOccurs=”0” maxOccurs=”unbounded”/>
9    </sequence>
10   </complexType>
11 </element>

12  <complexType name=” Tpublicacao”>
13  <sequence>
14 <element name=”ano” type=”String”/>
15 <sequence>
16    <element name=”isbn” type=”Integer”/>
17    <element name=”cdu” type=”Integer”/>
18   </sequence>
19   <element name=”titulo” type=”String”/>
20  </sequence>
21       <attribute name=”autoresref” type=”IDREFS” use=”required“/>
22 </complexType>

23 <complexType name=”Tartigo”>
24  <complexContent>
25   <extension base=”Tpublicacao” >
26    <sequence>
27     <element name=”local-publicacao” type=”String”/>
28    </sequence>
29   </extension>
30  </complexContent>
31 </complexType>

32 <complexType name=”Tlivro”>
33  <complexContent>
34   <extension base=”Tpublicacao”>
35    <sequence>
36     <element name=”editora” type=”String”/>
37    </sequence>
38   </extension>
39  </complexContent>
40 </complexType>

41 <complexType name=” Tautor”>
42  <sequence>
43   <element name=”nome” type=”String”/>
44   <element name=”e-mail” type=”String”/>
45  </sequence>
46  <attribute name=”id_autor” type=”ID” use=”required“/>
47       <attribute name=”instituicaoref” type=”IDREF” use=”optional“/>
48 </complexType>

49 <complexType name=” Tinstituicao”>
50  <sequence>
51   <element name=”nome” type=”String”/>
52          <element name=”telefone” minOccurs=”0” maxOccurs=”“/>
53   <element name=”endereco” type=”Tendereco”/>
54  </sequence>
55       <attribute name=”id_instituicacao” type=”ID” use=”required“/>
56 </complexType>

57 <complexType name=”Tendereco”>
58  <sequence>
59   <element name=”cidade” type=”String”/>
60   <element name=”estado” type=”String”/>
61   <element name=”pais” type=”String”/>
62  </sequence>
63 </complextype>

64 </Schema>
```

Figura 3.5: XML Schema de *bib.xml*

3.2.1.1 Declaração de Elemento

Elementos são declarados utilizando o marcador <element>, onde os atributos name e type definem seu nome e seu tipo, respectivamente. O tipo de um elemento pode também

ser especificado através de uma declaração de tipo anônima, como veremos na seção 4.2.1.3. Na declaração de um elemento também é possível definir a sua cardinalidade, através dos atributos minOccurs e maxOccurs. Por exemplo, a declaração <element name="autor" type="Tautor" minOccurs="0" maxOccurs="unbounded"/> (linha 7) especifica que o elemento secao é do tipo Tsecao e a restrição de ocorrência é opcional e multi-ocorrência.

3.2.1.2 Declaração de Atributo

Atributos são declarados utilizando o marcador <attribute>. Assim como na declaração de elementos, os atributos name e type determinam o nome e o tipo do atributo declarado, respectivamente. Entretanto, não é possível definir valores de mínimo e máximo de ocorrência para atributos, pois cada atributo pode ocorrer no máximo uma vez em um determinado elemento. No entanto, o atributo use do marcador <attribute> é usado para definir se a ocorrência de um atributo é opcional ou obrigatória. O atributo use também é usado para definir outros tipos de restrições como de valor fixo e de valor padrão. Por exemplo, a declaração <attribute name="id_instituicao" type="ID" use="required"/> (linha 55) especifica que este atributo é do tipo ID, monovalorado e obrigatório.

3.2.1.3 Definição de Tipos

Os tipos restringem o conteúdo permitido para um elemento ou atributo. Existem duas espécies de tipos em XML *Schema*: tipos simples (simpleType) e tipos complexos (complexType). 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.

Na linguagem XML *Schema*, para definir um tipo complexo, usamos o marcador <complexType>. Os tipos simples não precisam ser definidos, pois a especificação da XML *Schema* contempla um conjunto expressivo de tipos primitivos [62]. No entanto, é possível criar novos tipos de dados simples que correspondem a refinamentos de tipos primitivos. Para esse propósito, usamos o marcador <simpleType>.

O atributo name dos marcadores <simpleType> e <complexType> define o nome do tipo que está sendo definido. Quando esses marcadores não possuem o atributo name, dizemos que se trata de uma declaração de tipo anônima. Neste caso, a definição do tipo vem aninhada na declaração de um elemento. Por exemplo, no esquema da Figura 3.5, a declaração do elemento bib (linhas 2 a 11) contém uma definição de tipo anônima, que especifica que o elemento bib contém os elementos artigo, livro, autor e instituicao.

Nas declarações de tipos complexos, além das restrições de ocorrência declaradas para elementos individuais, a linguagem XML *Schema* provê restrições a serem aplicadas em grupos de elementos. Essas restrições também podem ser chamadas de delimitadores de grupos. Existem três tipos de delimitador de grupo:

- ***sequence***. Indica que os elementos do grupo devem aparecer no documento na mesma ordem em que foram declarados no esquema. Por exemplo, na Figura 3.5 (linhas 49 a 56), a declaração do tipo complexo Tinstituicao especifica que os elementos nome, telefone e endereco, contidos neste tipo, devem aparecer nesta ordem no documento.
- ***all***. Estabelece que todos os elementos do grupo podem aparecer uma ou nenhuma vez, e que eles podem aparecer em qualquer ordem;
- ***choice***. Indica que somente um dos elementos declarados no grupo pode aparecer no documento.

3.2.1.4 Representação Gráfica para XML *Schemas*

Nesta seção, apresentamos a notação utilizada neste trabalho para representar graficamente a estrutura definida por um XML *Schema*. Esta notação utiliza uma representação estruturada em "árvore de diretórios", onde cada nó da árvore representa um elemento ou atributo declarado no respectivo XML *Schema*.

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 (se não for anônimo) e sua restrição de ocorrência. Para atributos, seu nome é precedido do símbolo

"@". Se o elemento ou atributo é monocorrência e obrigatório, a restrição de ocorrência é vazia. Se o elemento for monocorrência e opcional, a restrição de ocorrência é "?". Se o elemento for multiocorrência e opcional, a restrição de ocorrência é "*". Finalmente, se o elemento for multiocorrência e obrigatório, a restrição de ocorrência é "+".

A árvore mostrada na Figura 3.6 é a representação gráfica do XML *Schema* apresentado na Figura 3.5. O primeiro nó da árvore é bib, e representa o elemento raiz bib declarado na linha 2 do XML *Schema*. Como podemos perceber no XML *Schema*, a declaração de tipo desse elemento é anônima. Sendo assim, sua representação gráfica não acompanha um tipo como nos demais nós da árvore. Os nós filhos de bib são T_{livro}, T_{artigo} , T_{autor} e $T_{instituicao}$, pois no XML *Schema* esses elementos estão declarados como o conteúdo do tipo complexo de bib (linhas 3 a 10 da Figura 3.5). Cada um desses nós filhos são rotulados com o nome do respectivo elemento, o símbolo "*", que denota a restrição de ocorrência zero ou muitos, e o nome do respectivo tipo.

3.3 Linguagens de Consulta para Documentos XML

Dados XML diferem de dados de banco de dados relacionais ou orientados a objetos em diversos aspectos, dentre estes podemos destacar o fato de que dados XML não seguem uma estrutura rígida como os esquemas relacionais e orientados a objetos. Assim, linguagens de consultas convencionais como SQL e OQL não são adequadas para definir consultas em documentos XML, portanto, são necessárias linguagens específicas para consultar esses documentos. Muitas linguagens de consulta para XML têm sido propostas, como, XML-QL [21], XSL [64], XQL [14], *XPath* [60] e *XQuery* [63].

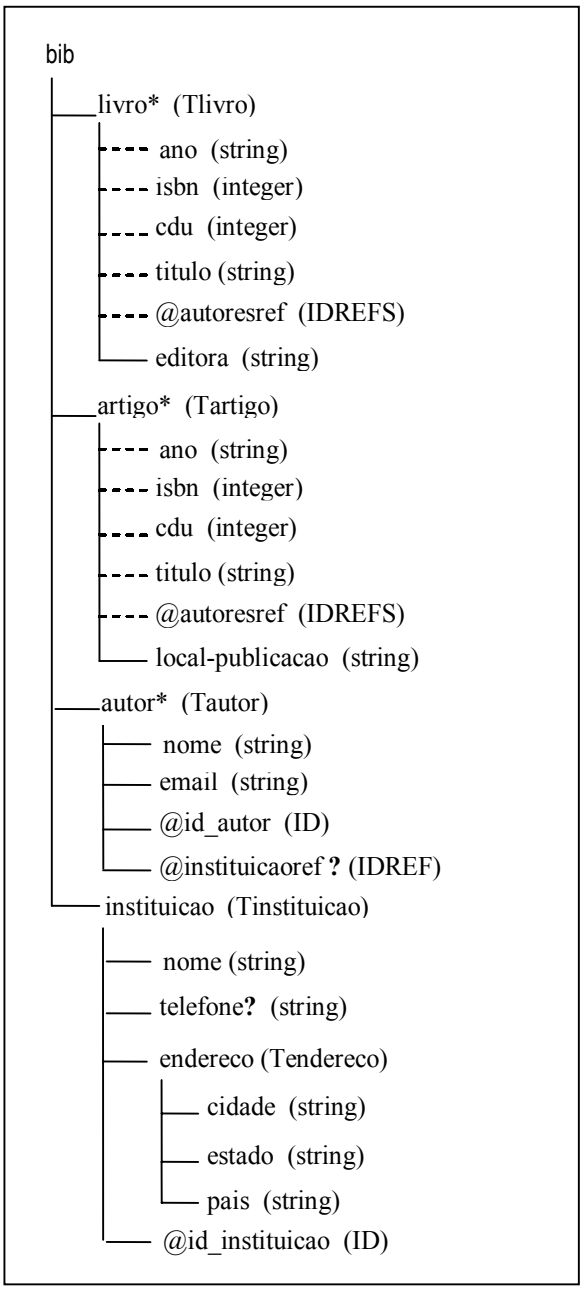


Figura 3.6: Representação gráfica para bib.xml.

Na abordagem de implementação das VCN's como visões XML, uma VCN pode ser definida através de consulta *XQuery* sobre a visão XML da aplicação, como veremos no Capítulo 5. *XQuery* usa *XPath* para endereçar partes do documento XML consultado. Sendo assim, descrevemos a seguir os principais aspectos das linguagens *XPath* e *XQuery* propostas pelo W3C.

3.3.1 *XPath*

A linguagem *XPath* é uma recomendação do W3C para acessar partes de um documento XML. Esta linguagem usa uma notação de caminhos para navegar através da estrutura hierárquica de um documento XML e foi projetada para ser inserida em outras linguagens chamadas de *host languages*, tais como XSLT e *XQuery*.

XPath modela um documento XML como uma árvore de nós. Este modelo em árvore é apenas conceitual e possui diferentes tipos de nós, tais como, nós-documento, nós-elemento, nós-texto, nós-atributo, entre outros. Para cada tipo de nó, há um modo de determinar seu valor literal (*string-value*). Para alguns tipos de nós o valor é parte do nó; para outros, esse valor é computado a partir do valor de nós descendentes. Por exemplo, o valor de um nó-elemento consiste da concatenação do valor de todos os nós-texto descendentes do nó-elemento na ordem do documento.

O nó-documento encapsula todo o documento XML; ele é o ponto de partida na árvore que descreve o documento XML. Os nós-elemento encapsulam os elementos XML, e podem ter nós-elemento, nós-texto e outros como seus filhos. Nós-elementos também podem conter nós-atributos.

O bloco de construção básico da *XPath* é a expressão. A linguagem provê diversos tipos de expressão que podem ser construídos a partir de palavras-chave, símbolos e operandos. Em geral, os operandos de uma expressão são outras expressões. *XPath* é uma linguagem funcional que permite que vários tipos de expressão sejam aninhados.

A expressão que localiza um nó em uma árvore e retorna uma seqüência de nós distintos na ordem do documento é chamada de expressão de caminho. 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.

A seguir, exemplificamos algumas expressões de caminho utilizadas para consultar o documento XML bib.xml apresentado na Figura 3.4.

Exemplo 3.1

A expressão de caminho **document("bib.xml")/bib/artigo/titulo** consiste em quatro passos: o primeiro passo seleciona o documento XML bib.xml; o segundo seleciona o elemento raiz de bib.xml; o terceiro seleciona os elementos artigo, filhos do elemento raiz e o quarto passo seleciona os elementos titulo dos elementos artigo recuperados no passo anterior. Assim, essa expressão retorna todos os elementos titulo contidos nos elementos artigo, contidos no elemento bib do documento bib.xml. A figura abaixo mostra o documento XML retornado por essa consulta:

```
<titulo> Temporal Serialization Graph Testing </titulo>
```

Para simplificar os próximos exemplos, utilizamos a variável **$bib** para substituir a expressão **document**("bib.xml")/bib:

Exemplo 3.2

A expressão de caminho **$bib/autor/nome/text()** retorna o nome de todos os elementos autor contidos em **$bib** (Ângelo Brayner e José Maria Monteiro).

Exemplo 3.3

A expressão de caminho **$bib/livro [ano = "1999"]** retorna todos os elementos livro em **$bib** que possuem o elemento ano igual a "1999". A figura abaixo mostra o XML retornado por essa consulta:

```
<livro autoresref="A">
    <ano> 1999 </ano>
    <isbn> 3826561422 </isbn>
    <titulo> Transaction Management in Multidatabase Systems </titulo>
    <editora> Shaker-Verlag </editora>
 </livro>
```

3.3.2 *XQuery*

XQuery [63] é a linguagem de consulta padrão que está sendo elaborada pelo W3C e utiliza o mesmo modelo de dados apresentado na seção anterior. Trata-se de uma linguagem funcional na qual consultas são representadas como expressões. Numa consulta *XQuery* expressões têm a forma de variáveis, expressões de caminhos, chamada a funções, expressões condicionais, construtores de elementos, expressões FLWR etc., e podem ser aninhadas e combinadas usando operadores lógicos e aritméticos.

Para navegar através de caminhos na estrutura de um documento XML, *XQuery* usa expressões de "caminho" cuja sintaxe é uma abreviação da sintaxe de *XPath*. O resultado do cálculo de uma expressão de caminho sobre um documento XML retorna uma floresta ordenada consistindo em nós que satisfazem a expressão e seus descendentes [9]. A ordem do resultado é ditada pela ordem dos elementos dentro do documento XML consultado.

Uma característica poderosa de *XQuery* é a presença de expressões FLWR (*for-let-where-return*). Uma expressão FLWR é usada sempre que é necessário interagir sobre elementos de uma coleção. As cláusulas *for-let* fazem variáveis interagirem sobre o resultado de uma expressão ou atribui o valor de expressões arbitrárias a variáveis. A cláusula *where* permite especificar restrições sobre essas variáveis. A cláusula *return* gera a saída da expressão

```
<?xml version="1.0"?>
<livros>
        <livro isbn="073571020">
                <titulo>Inside XML</titulo>
                <editora>New Riders</editora>
                <autores>
                        <autor>Steven Holzner</autor>
                </autores>
        </livro
        <livro isbn="8535206485">
                <titulo>Gerenciando Dados na Web</titulo>
                <editora>Campus</editora>
                <autores>
                        <autor>Serge Abiteboul</autor>
                        <autor>Peter Buneman</autor>
                        <autor>Dan Suciu</autor>
                </autores>
        </livro>
</livros>
```

Figura 3.7: Documento liv.xml.

FLWR, que pode ser um nodo, uma floresta ordenada de nodos, ou um valor primitivo. A cláusula *return* contém uma expressão geralmente composta por construtores de elementos, variáveis e sub-expressões aninhadas. A seguir, apresentamos alguns exemplos de consultas *XQuery* para o documento XML liv.xml apresentado na Figura 3.7.

Exemplo 3.4

A consulta abaixo retorna os títulos de todos os livros. Essa consulta é composta por uma expressão *for-return*. A cláusula **for** especifica uma iteração sobre os elementos livro que são filhos do elemento raiz livros no documento XML liv.xml (função de entrada **document**). Em cada iteração, o respectivo elemento livro será armazenado na variável **$i**. A cláusula **return** especifica que a cada iteração do **for**, o conteúdo textual (função **text**()) do elemento titulo de **$i** deverá ser mostrado:

Consulta	Resultado
for $i in document("liv.xml")/livros/livro **return** **$i**/titulo/**text**()	Inside XML Gerenciando Dados na Web

Figura 3.8: Consulta *XQuery* do Exemplo 3.4.

Exemplo 3.5

A consulta abaixo retorna todos os autores do livro "Inside XML". Essa consulta é composta por uma expressão *for-where-return*. A cláusula **For** especifica uma iteração sobre os elementos livro que são filhos do elemento raiz livros no documento XML liv.xml. Em cada iteração, o respectivo elemento livro será armazenado na variável $i. A presença da cláusula **where** especifica que a cláusula **return** será executada apenas quando o titulo do livro for igual a "Inside XML". A cláusula **return** especifica que a cada iteração válida do **For**, o sub-elemento autores de $i deverá ser mostrado.

Consulta	Resultado
For $i in **document**("liv.xml")/livros/livro **where** $i/titulo = "Inside XML" **return** **$i**/autores	<autores> <autor>Steven Holzner</autor> </autores> <autores> <autor>Serge Abiteboul</autor> <autor>Peter Buneman</autor> <autor>Dan Suciu</autor> </autores>

Figura 3.9: Consulta *XQuery* do Exemplo 3.5.

Exemplo 3.6

Para cada livro, criar um elemento <livro_autores> contendo o isbn do livro e um sub-elemento <nome_autor> para cada autor desse livro. Esse exemplo enfatiza o uso de construtores de elementos (<livro_autores> e <nome_autor>) e de expressões *for-where-return* aninhadas.

Consulta	Resultado
for $i** in **document**("liv.xml")/livros/livro **return** <livro_autores>{ **$i/@isbn/text(), **for $j in $i**/autores **return** <nome_autor>**{** **$j**/autor/text() **}**</nome_autor> }</livro_autores>	<livro_autores> 0735710201 <nome_autor>Steven Holzner</nome_autor> </livro_autores> <livro_autores> 8535206485 <nome_autor>Serge Abiteboul</nome_autor> <nome_autor>Peter Buneman</nome_autor> <nome_autor>Dan Suciu</nome_autor> </livro_autores>

Figura 3.10: Consulta *XQuery* do Exemplo 3.6.

3.4 Transformando Documentos XML usando XSLT

XSLT (*eXtensible Stylesheet Language for Transformations*) [64] é uma linguagem poderosa e flexível para transformar um documento XML em um outro documento

(documento de saída), que pode ser um documento HTML, outro documento XML, um arquivo PDF, um arquivo SVG, um código Java, um arquivo texto, etc [51].

Um estilo XSL é um tipo particular de documento XML, que contém regras de transformação, chamadas *templates*. Baseado nas regras do estilo e em um dado documento XML de origem, o processador XSLT gera o documento de saída transformado. XSLT é uma recomendação oficial do W3C, cujas principais características são [7]:

- Regras declarativas (não-procedurais) podem ser adicionadas ou removidas em qualquer ordem sem afetar as outras regras;
- Regras de transformação podem reordenar elementos do documento origem durante o processo de transformação do documento de saída;
- Regras de transformação podem filtrar elementos e atributos do documento origem;
- Regras de transformação não podem modificar o documento de origem, nem o resultado produzido por outras regras.

Durante a transformação de um documento, o processador XSLT converte o documento XML de entrada em uma estrutura de árvore cujos nós correspondem aos elementos e atributos do documento XML. De forma similar, o estilo XSLT é também convertido em uma estrutura de árvore. À medida que as regras de transformação do estilo vão sendo aplicadas ao documento XML de entrada pelo processador, é gerada uma estrutura de árvore, que corresponde ao documento de saída.

O estilo XSLT apresentado na Figura 3.11 ilustra as regras para transformar o documento XML da Figura 3.7 no documento HTML mostrado na Figura 3.12. A cláusula <xsl:template match="/livros/livro"> define uma regra, onde o atributo match define o nó na árvore do documento de origem a ser transformado. Esse *template* pode conter elementos ou outras regras XSLT, e pode ser parametrizado. Por exemplo, na linha 3, a *tag* <TITLE> contém uma regra (<xsl:value-of select = "titulo">) que extrai o título do livro do documento XML de entrada. Assim, o documento HTML de saída apresenta o título dos livros. Na linha 10, a cláusula <xsl:attribute > define um atributo href na tag <a> que indica um link para o documento XML livro.xml, passando o isbn do livro selecionado como parâmetro.

A seguir, apresentamos os principais elementos da especificação XSLT.

- **<xsl:attribute>**: esse elemento permite criar um atributo no documento de saída. O atributo name do elemento define o nome do atributo criado.

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3.  <xsl:template match="/livros/livro">
4.  <html>
5.      <head>
6.      <title><xsl:value-of select="titulo"/></title></head>
7.      <body>
8.          <center>
9.              <a>
10.                 <xsl:attribute name="href">livro.xml?isbn=
11.                         <xsl:value-of select="@isbn"/>
12.                 </xsl:attribute>
13.                 <xsl:value-of select="titulo"/>
14.             </a>
15.         </center>
16.         ISBN:<xsl:value-of select="@isbn"/>
17.         <br></br>
18.         Editora: <xsl:value-of select="editora"/>
19.         <br></br>
20.         Autores:
21.         <xsl:for-each select="autores/autor">
22.             <xsl:value-of select="."/>
23.             <br></br>
24.         </xsl:for-each>
25.     </body>
26. </html>
27. </xsl:template>
28.</xsl:stylesheet>
```

Figura 3.11: Estilo XSLT com regras de transformação.

- **<xsl:element>**: similar ao elemento <xsl:attribute>, <xsl:element> permite criar um elemento no documento saída. O atributo name do elemento define o nome do elemento criado.
- **<xsl:copy-of>**: esse elemento copia toda a árvore de um elemento no documento de entrada na árvore do documento de saída. O atributo select define o nó a ser copiado.
- **<xsl:for-each>**: esse elemento age como um operador de interação XSLT. Esse elemento tem um atributo select que seleciona os nós, e é avaliado para cada nó selecionado.
- **<xsl:param>**: define um parâmetro para o template. O nome do parâmetro é definido pelo atributo name.
- **<xsl:stylesheet>**: esse elemento é geralmente o elemento raiz de um estilo XSLT, e define a versão da linguagem XSLT utilizada com a definição da *namespace* XSL (http://www.w3.org/XSL/Transform)

- **<xsl:template>**: esse elemento define uma regra para transformar parte de um documento XML de entrada no documento de saída. A atributo match define o elemento a ser transformado.
- **<xsl:value-of>**: calcula o valor de uma expressão *XPath*, converte o valor para string e escreve o valor na árvore do documento de saída. O atributo *select* define a expressão *XPath* a ser avaliada e escrita no documento de saída.

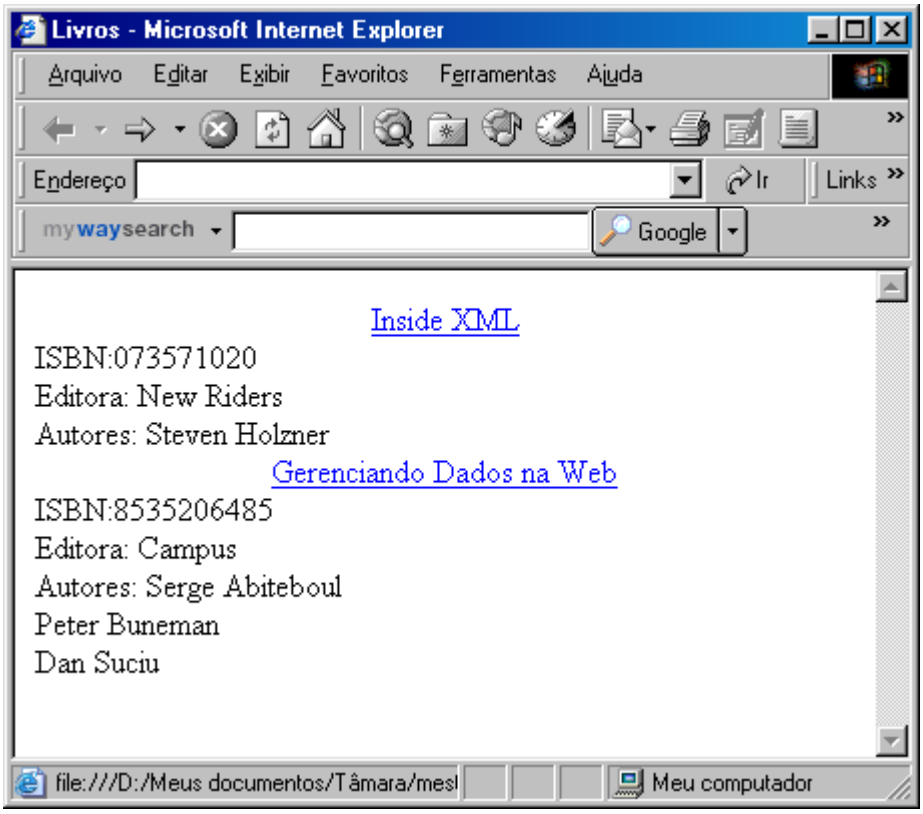


Figura 3.12: Documento HTML resultante da transformação.

3.5 *Frameworks* para Publicação de Dados Armazenados em Banco de Dados Objeto-relacionais como XML

A linguagem XML vem se tornando padrão para integração, publicação e troca de dados entre as aplicações na Web. Para facilitar a troca de dados, grupos de empresas definem esquemas XML públicos que especificam o formato dos dados XML a serem compartilhados entre suas aplicações. Como a maioria dos dados está armazenada em

bancos de dados relacionais ou baseados em objetos, surgiu a necessidade de se definir mecanismos para publicá-los no formato XML. Uma forma geral de se publicar esses dados é feita através do uso de visões XML, que podem ser consultadas por aplicações/usuários *web* utilizando linguagens de consultas próprias para XML, tais como *XML*-QL [21], *XPath* [60], *XQuery* [63] etc. Visões XML devem suportar consultas, pois, geralmente, uma aplicação *web* tem interesse apenas em uma parte dos dados publicados através da visão [47].

Diversos sistemas foram desenvolvidos para permitir a publicação de dados no formato XML a partir de dados armazenados em bases convencionais [6][46][47][19][20][42]. Esses sistemas funcionam como *middleware* entre a aplicação e o banco de dados. Visões XML são definidas e consultas *XQuery* formuladas sobre essas visões são traduzidas pelo sistema em consultas SQL. Assim, todo processamento das consultas SQL é atribuído ao processador do banco de dados. O resultado da consulta é retornado pelo sistema no formato XML.

Outra forma de publicar dados no formato XML é com o uso da tecnologia XSQL *Pages*. O *Oracle XSQL Pages Publishing Framework* [34] é um *framework* que dá suporte à geração dinâmica de XML na *web* a partir de dados objeto-relacionais. Uma página XSQL define uma consulta SQL:1999, sobre um banco de dados objeto-relacional, que é processada a cada requisição *web* e seu resultado convertido em um documento XML no formato Canônico.

Nesta seção, apresentamos algumas das propostas para publicação de dados relacionais e objeto relacionais através de visões XML, e também o *framework* XSQL *Pages*.

3.5.1 XPeranto

O *XPeranto* (XML *Publishing of Entities*, *Relashionships*, *And Typed Objects*) [6][46][47] é um sistema que funciona como *middleware* entre aplicações e SGBD's relacionais, permitindo a essas aplicações manipularem os dados dos SGBD's como visões XML, a partir de consultas *XQuery*. Consultas *XQuery* submetidas ao *XPeranto*, são traduzidas em consultas SQL sobre o banco de dados.

Inicialmente, o *XPeranto* disponibiliza uma Visão XML *Default* do banco de dados existente, a qual especifica a estrutura do banco de dados. Nessa visão, **db** é o elemento raiz. Cada tabela no banco de dados é um elemento filho de **db**. Elementos nomeados por **row** são colocados dentro de cada elemento da tabela. Dentro de um elemento **row**, os nomes das colunas das tabelas aparecem como marcadores e o valor das colunas aparece como texto. A Figura 3.14 mostra a Visão XML *Default* gerada a partir do banco de dados mostrado na Figura 3.13.

Secao_rel

id	descrição
01	Banco de Dados
02	Redes de Computadores

Autor_rel

id	nome	foto	biografia
10	Francisco		...
11	João		...

Materia_rel

secid	título	resumo	data_pub	autid
01	BDs XML nativo	...	05/05/03	10
01	BDOR	...	10/10/03	10

Figura 3.13: Esquema do Banco de Dados

```
<db>
        <secao_rel>
                <row><id>01</id><descricao>Banco de Dados</descricao></row>
                <row><id>02</id><descricao>Redes de Computadores</descricao></row>
        </secao_rel>
        <materia_rel>
                <row><secid>01</secid><titulo>BDs XML nativo</titulo>
                        <resumo>…</resumo><data_pub>05/05/03</data_pub><autid>10</autid>
                </row>
                <row><secid>01</secid><titulo>BDOR</titulo>
                        <resumo>…</resumo><data_pub>10/10/03</data_pub><autid>10</autid>
                </row>
        </materia_rel>
        <autor_rel> ...  </autor_rel>
</db>
```

Figura 3.14: Visão XML Default

Em seguida, usuários podem definir suas próprias visões no topo da visão XML *Default* ou simplesmente submeter consultas *XQuery* ao sistema. Uma visão XML é definida através de uma consulta *XQuery* que mapeia a visão XML *Default* na visão XML desejada. A Figura 3.15 mostra a Visão XML Seções definida sobre a visão XML *Default* do banco de dados.

```
create view secoes as (
<raiz>{
for $secao in view("default")/secao_rel/row
return
    <secao>{
        <nome>{$secao/descrição}</nome>
        for $materia in view("default")/materia_rel/row
        where $secao/id = $materia/secid
        return
            <materia>{
                <titulo>{$materia/titulo}</titulo>,
                <data_pub>{$materia/data_pub}</data_pub>,
                <resumo>{$materia/resumo}</resumo>,
                let $autor in view("default")/autor_rel/row
                where $autor/id=$materia/autid
                return
                    <autor>{
                        <nome>{$autor/nome}</nome>
                        <foto>{$autor/foto}</foto>
                        <biografia>{$autor/biografia}</biografia>
                    }<autor>
            }</materia>
    }</secao>
}</raiz>
```

Figura 3.15: Visão XML Seções

O processamento das consultas *XQuery* submetidas ao *XPeranto*, por exemplo, é mostrado na Figura 3.16. Quando uma consulta *XQuery* é submetida ao *framework*, ela é convertida para uma representação interna chamada XML *Query Graph Model* (XQGM) pelo módulo *XQuery Parser*. O módulo *Query Rewrite & View Composition* realiza uma composição entre a consulta XQGM e a Visão XML sobre a qual a consulta é definida, e executa otimizações para eliminar a construção de fragmentos desnecessários. O módulo *Computation Pushdown* trata da tradução da consulta XQGM modificada em uma consulta SQL sobre o banco de dados e, além disso, gera uma estrutura chamada *Tagger Graph*, que captura a estrutura da consulta XQGM. Essa estrutura é utilizada pelo módulo *Tagger Runtime* para construir o documento XML com o resultado final da consulta.

Recentemente, a IBM tornou o *XPeranto* uma ferramenta comercial chamada IBM *For Tables* [26], capaz de processar consultas *XQuery* sobre visões XML de banco de dados DB2.

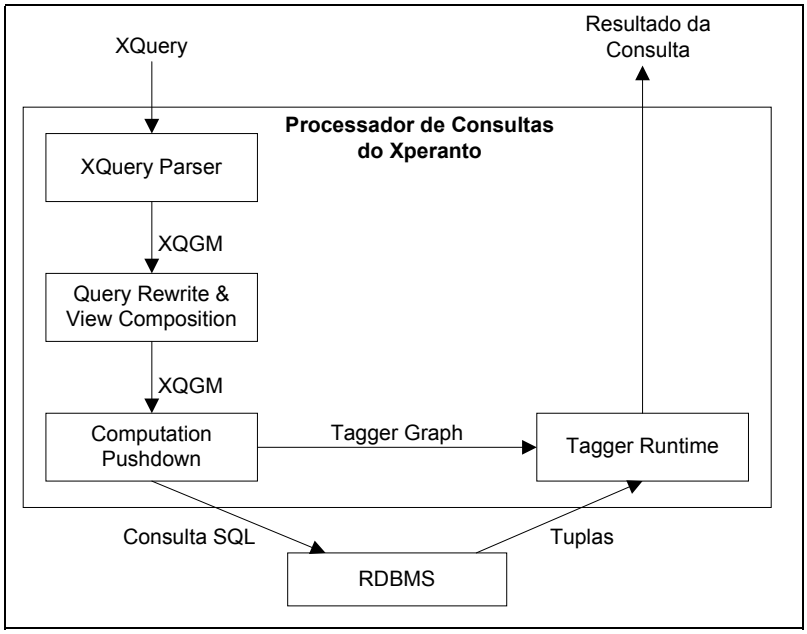


Figura 3.16: Arquitetura do *Xperanto*

3.5.2 Silkroute

Silkroute [19][20] é um *framework* para publicação de dados armazenados em um banco de dados relacional no formato XML. Assim como o *XPeranto*, o *Silkroute* serve como *middleware* entre um banco de dados relacional e uma aplicação que acessa esses dados na *web*. O processo de publicação de dados no formato XML no *Silkroute* é feito de forma similar ao *XPeranto*. No *Silkroute*, esse processo é feito através dos diversos passos discutidos a seguir.

Primeiro, uma visão XML canônica virtual do banco de dados é gerada automaticamente a partir do esquema do banco de dados. Uma visão XML Canônica no *Silkroute* é muito parecida a uma visão XML *Default* do *Xperanto* e permite ao projetista especificar consultas *XQuery* sobre os dados relacionais.

Em seguida, o projetista especifica uma consulta pública em *XQuery* sobre a visão XML canônica, definindo assim a *visão* XML pública. As aplicações têm acesso somente à visão XML pública e não ao banco de dados.

Para acessar os dados do banco de dados, um usuário/aplicação submete uma consulta *XQuery* sobre a visão XML pública, extraindo somente os dados de interesse da aplicação. O processamento das consultas *XQuery* submetidas ao *Silkroute* é mostrado na Figura 3.17. O módulo *View Forest Composer* do sistema realiza uma composição da consulta da aplicação com a consulta que define a visão XML pública, resultando em uma nova consulta. Essa nova consulta é submetida ao módulo *Planner*, o qual a traduz em uma ou mais consultas SQL equivalentes. Em geral, essa tradução pode produzir diversos conjuntos de consultas SQL que produzem o mesmo resultado da consulta *XQuery*. Esses conjuntos de consultas SQL são chamados de plano de execução. Com o objetivo de selecionar um bom plano de execução de consultas, o módulo *Planner* recebe também como entrada algumas estimativas de custo de consultas produzidas pelo SGBD. Além da tradução, o módulo *Planner* gera um XML Template para construção do documento XML de resposta.

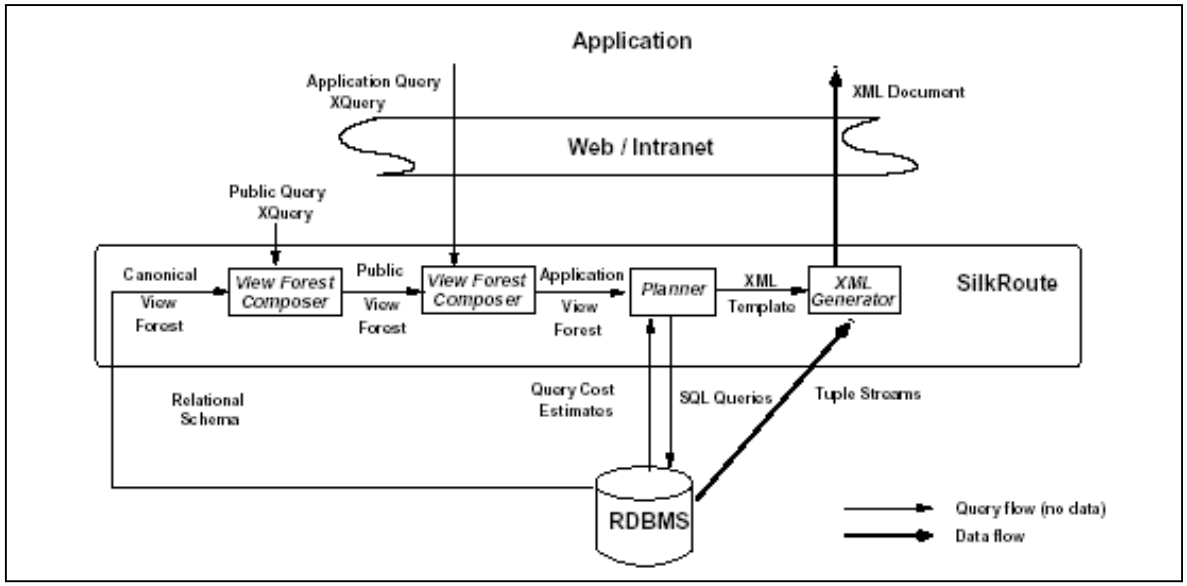


Figura 3.17: Arquitetura do *Silkroute*.

O conjunto de consultas SQL obtido é submetido ao SGBD, que realiza todo o processamento das consultas. As tuplas resultantes das consultas são retornadas para o Silkroute e são manipuladas no módulo *XML Generator*. A partir dessas tuplas e do XML Template, o módulo *XML Generator* produz um documento XML com o resultado das consultas, o qual é retornado para a aplicação.

Internamente, o Silkroute utiliza uma abstração chamada *View Forest* para representação das expressões *XQuery*. A instância do banco de dados relacional (visão canônica), a visão pública e as consultas submetidas ao *framework* são representadas por uma View Forest, que define um mapeamento de um banco de dados relacional para um documento XML e separa a estrutura do documento XML de saída do processamento que produz o conteúdo do documento. A estrutura é representada por uma ou mais árvores cujos nós são rotulados com os nomes dos elementos XML, nomes de atributos ou tipos atômicos. O processamento é representado por consultas SQL que são relacionadas aos nós. Nenhum valor XML é construído na View Forest, somente o resultado final da consulta SQL. Como exemplo, considere a figura abaixo que mostra o fragmento da View Forest da visão XML canônica que corresponde à tabela Clothing de um banco de dados qualquer:

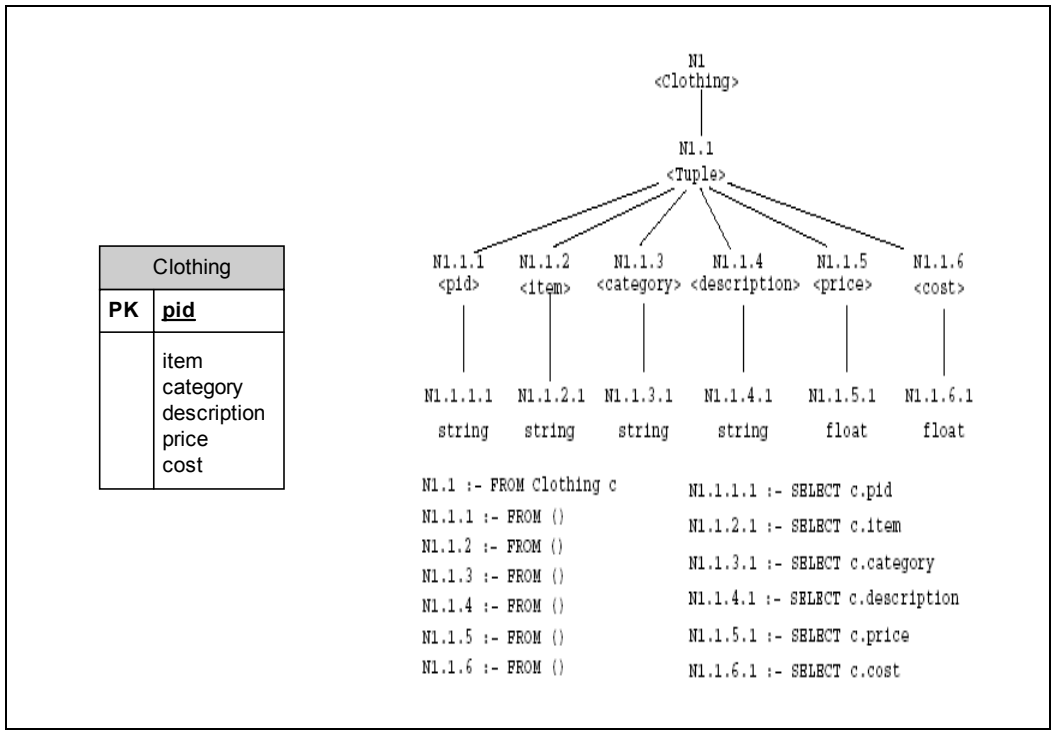


Figura 3.18: Fragmento da *View Forest* Canônica para a tabela *Clothing*.

3.5.3 XML *Publisher*

Em [42], é proposto o XML *Publisher*, um *framework* para publicação de dados objeto-relacionais através de visões XML. Para cada visão XML publicada no XML *Publisher* existe uma visão de objetos associada, chamada de visão de objetos *default* (VOD), onde a estrutura dos seus objetos é isomórfica à estrutura dos elementos da respectiva visão XML Como pode ser visto em [42], isso facilita a tradução das consultas XQuery sobre a visão XML em consultas SQL sobre o esquema do banco de dados.

As visões XML publicadas no XML *Publisher* podem ser consultadas e atualizadas usando a linguagem *XQuery* [63] Inicialmente, a consulta *XQuery* é traduzida em uma consulta SQL:1999 aplicada à respectiva VOD. Essa tradução é feita com base no esquema da visão XML. Em seguida, a consulta SQL:1999 gerada é processada pelo SGBD, com base na definição da VOD. Dessa forma, toda a complexidade do processamento dos dados é delegada ao SGBD. Finalmente, o resultado da consulta SQL:1999, que é uma coleção de objetos, deverá ser manipulado no XML *Publisher* para gerar a resposta XML especificada na consulta *XQuery* de entrada. Essa transformação de coleção de objetos para XML é feita com base na consulta *XQuery* e no esquema da visão XML. A Figura 3.19 explica como é processada cada consulta *XQuery* submetida ao XML *Publisher*:

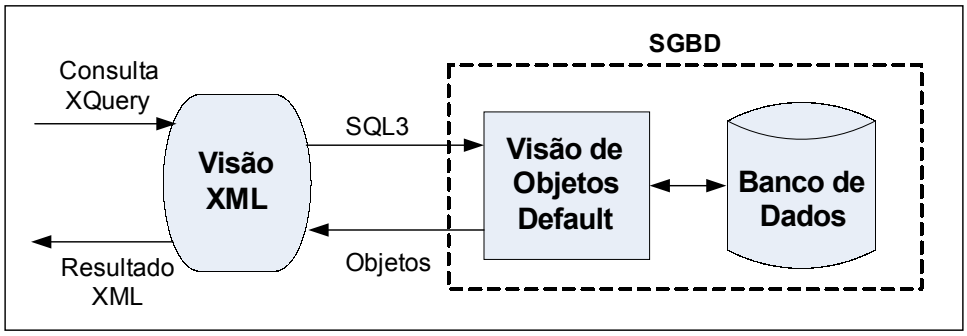


Figura 3.19: Tradução de Consultas no XML *Publisher*.

Os passos para o processamento de uma consulta *XQuery* aplicada a uma visão XML no XML *Publisher* são apresentados na Figura 3.20 e descritos a seguir:

i. O *XQueryTranslator* traduz a consulta *XQuery* em uma consulta SQL:1999 aplicada a respectiva VOD. Dado que a estrutura dos objetos da VOD tem a mesma estrutura dos elementos da visão XML, a tradução é simples e pode ser feita com base apenas no esquema da visão XML. Em seguida, a consulta SQL gerada é submetido ao SGBD.

ii. A consulta SQL é processada pelo SGBD com base na definição da visão de objeto. Dessa forma, toda a complexidade do processamento dos dados é delegada ao SGBD. O resultado desse processamento é um conjunto de objetos.

iii. Os objetos resultantes do passo "ii" são transformados pelo *XQueryTranslator* em um resultado XML com base na consulta XQuery original e no esquema da visão XML. Esse é o único passo no qual o *XQueryTranslator* realmente executa alguma manipulação de dados. Mas essa manipulação consiste apenas em percorrer a estrutura dos objetos resultantes para adicionar marcadores (elementos).

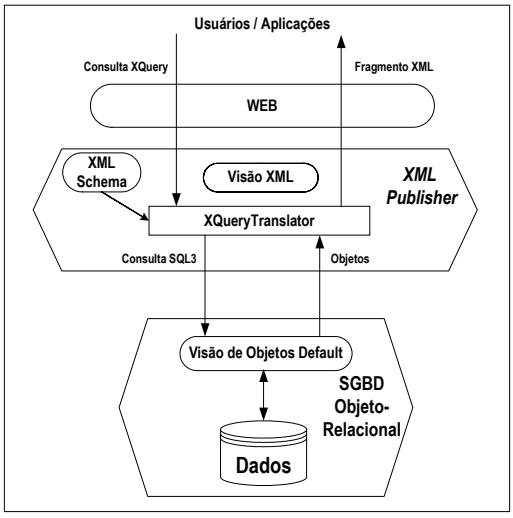


Figura 3.20: Processamento de consultas no XML *Publisher*

3.5.4 XSQL *Pages Publishing Framework*

A tecnologia XSQL *pages* (páginas XSQL), desenvolvida pela Oracle, combina o poder do XML, XSLT e XSQL para publicar conteúdo dinâmico na *web* no formato XML. Uma página XSQL é um documento XML processado pelo s*ervlet* XSQL [34], o qual possui consultas SQL parametrizadas que extraem informações de bancos de dados objeto-relacionais.

Considere, por exemplo, o banco de dados da Figura 3.21 e a página XSQL da Figura 3.22 (V3.xsql), a qual publica o resultado da consulta SQL contida na *tag* <xsql:query> no formato do esquema XML mostrado na Figura 3.23 (b). O resultado da consulta SQL é transformado pelo processador XSQL em um documento XML no formato canônico. A Figura 3.23 (a) mostra o esquema XML Canônico do documento XML retornado pelo XSQL para a consulta em V3.xsql. A Figura 3.24 mostra um exemplo de um documento no formato canônico resultado do processamento da página XSQL da Figura 3.22.

Com o uso da tecnologia XSLT, a estrutura do documento XML canônico pode ser alterada de acordo com o requerido pelo esquema XML da aplicação. Para isso, na página XSQL da Figura 3.22, o atributo href associa um documento stylesheet nomeado V3.xsl, que utiliza uma folha de estilo XSLT para transformar o documento XML canônico no formato do esquema XML da aplicação (Figura 3.23(b)). A correspondência entre os elementos do esquema XML canônico e os elementos do esquema XML da aplicação também é mostrada na Figura 3.23. A Figura 3.24 mostra um exemplo das transformações aplicadas a um resultado da consulta SQL:1999 na página da Figura 3.22.

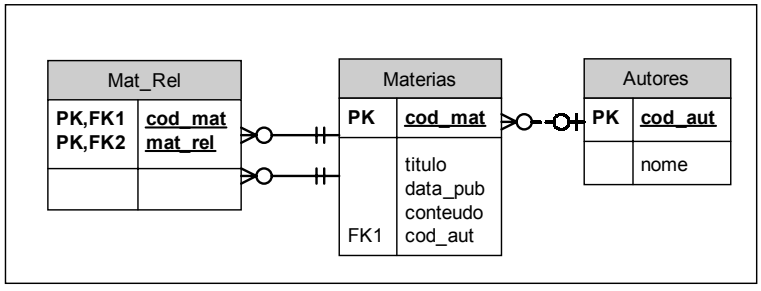


Figura 3.21: Banco de Dados Matérias Relacionadas.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="V3.xsl"?>
<xsql:query connection="xml" xmlns:xsql="urn:oracle-xsql">
        SELECT m.titulo as titulo,  m.data_pub as data_pub,  m.conteudo as conteudo,
                ( SELECT a.nome
                  FROM Autores a
                  WHERE m.cod_aut = a.cod_aut) as autor,
                  CURSOR ( SELECT m2.titulo, m2.data_pub
                           FROM Materias m2, Mat_Rel mr
                           WHERE mr.mat_rel = m2.cod_mat
                                 AND mr.cod_mat = m.cod_mat ) as mat_rel
        FROM Materias m
        WHERE m.titulo = {@titulo}
</xsql:query>
```

Figura 3.22: Página XSQL V3.xsql.

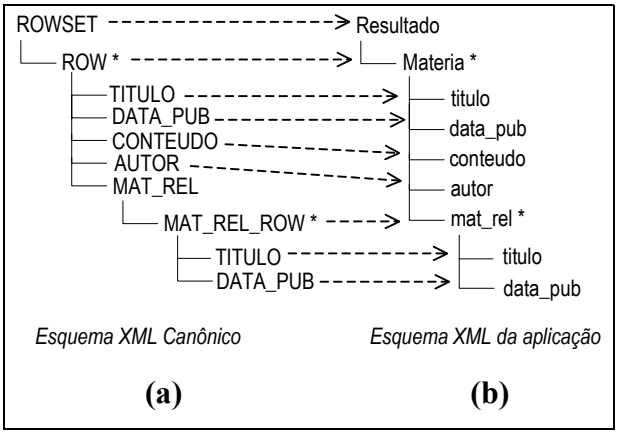


Figura 3.23: Esquemas XML para V3.xsql.

Uma página XSQL é mais versátil quando seus resultados podem mudar baseado nos valores de um ou mais parâmetros passados com a requisição [32]. Observe a expressão {@titulo} na cláusula WHERE da consulta SQL da página XSQL apresentada na Figura 3.22. Essa expressão denota um parâmetro de nome titulo, que corresponde ao título da matéria a qual se deseja obter seus detalhes. A cada requisição a essa página, a expressão {@titulo} será substituída pelo valor do parâmetro titulo passado na requisição.

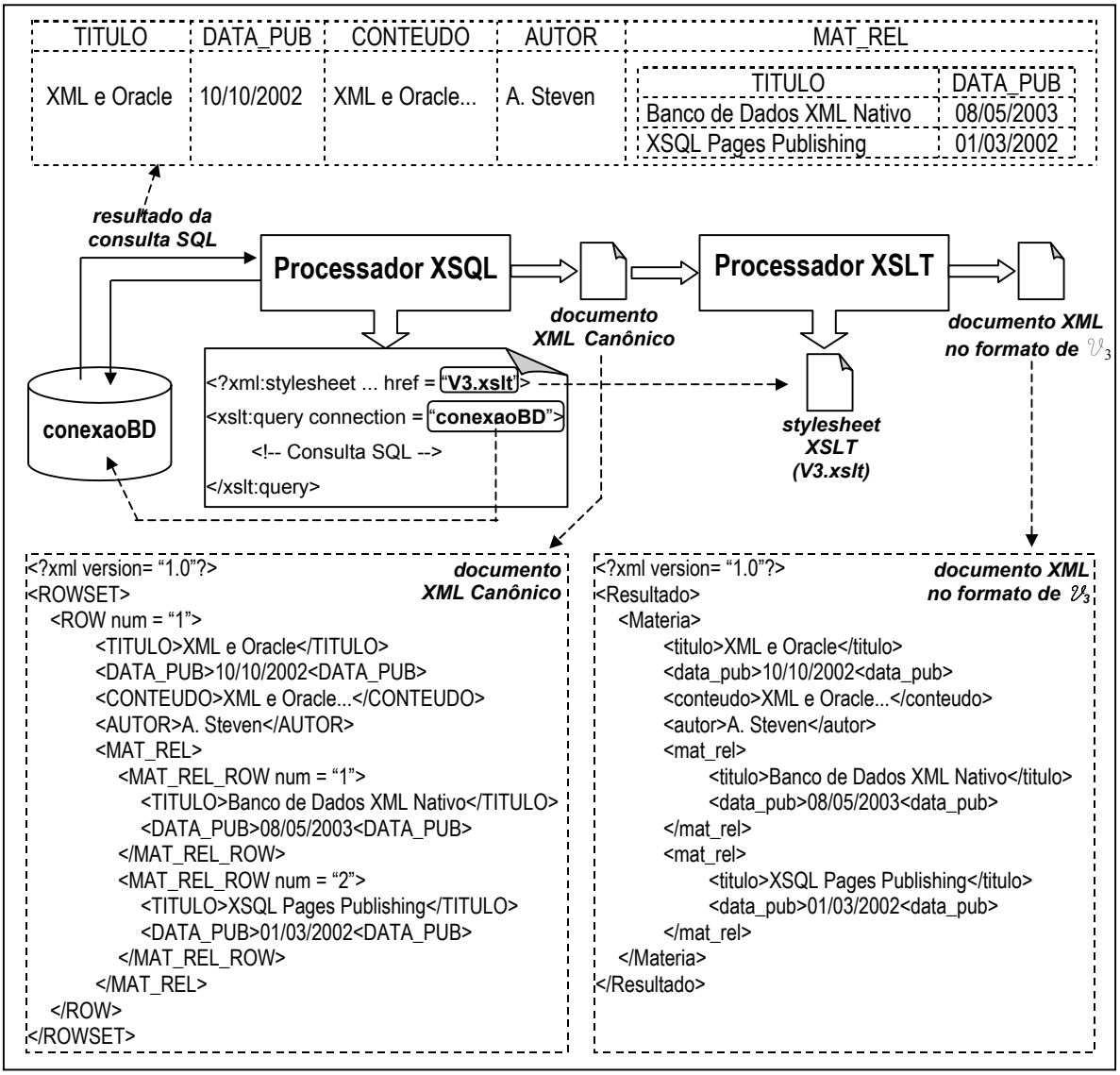


Figura 3.24: Publicação de dados XML baseado em consultas SQL usando XSQL *Pages*.

Capítulo 4
Projeto das Visões de Objeto e das Visões de Contexto de Navegação

Neste Capítulo, discutimos as atividades de Projeto das visões de objeto e de Projeto das Visões de Contexto de Navegação (VCN's). Na Seção 4.1, introduzimos os Diagramas de Interação dos Usuários (UID) [50][25], utilizados para representar graficamente as VCN's e as interações entre usuários e a aplicação. Na Seção 4.2, discutimos o projeto das visões de objeto a partir da integração das VCN's dos UID's. Na Seção 4.3, apresentamos a atividade de Projeto das VCN's, as quais são especificadas sobre as visões de objeto.

4.1 Diagrama de Interação dos Usuários e Visão de Contexto de Navegação

O desenvolvimento de uma aplicação é baseado nas informações coletadas durante a atividade Levantamento de Requisitos. Os requisitos de dados e funcionais que motivaram o desenvolvimento da aplicação são obtidos com base nas informações dos usuários e em documentação existente. Esses requisitos devem ser especificados da forma mais completa possível. Para especificação das tarefas a serem realizadas pela aplicação, a UML fornece cenários e diagramas de caso de uso. De acordo com [44], a atividade Levantamento de Requisitos consiste nos seguintes passos:

- Identificação dos autores e tarefas a serem realizadas através da aplicação;

- Especificação dos cenários, que descrevem os passos da interação do sistema com o usuário para a realização de cada tarefa;
- Especificação dos casos de uso, obtidos a partir do agrupamento dos cenários que descrevem a mesma tarefa.

Diagramas de Interação dos Usuários (UID) são utilizados para representar as interações entre usuários e a aplicação durante a realização de uma tarefa descrita por um caso de uso. Os UID's foram propostos para suprir as deficiências encontradas no emprego de casos de uso. Por serem descrições textuais, os casos de uso costumam causar ambigüidades e, assim, pecam 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.

Para cada caso de uso obtido é especificado um UID. Em [50] é apresentado como mapear um caso de uso no UID correspondente. É importante ressaltar a exigência de uma constante participação dos usuários da aplicação na validação dos casos de uso e UID's obtidos, a fim de garantir que a aplicação final atenda aos requisitos dos usuários.

Considere, por exemplo, a tarefa "*Ler matérias de uma seção"*, cujos caso de uso e UID são mostrados na Figura 4.1. Cada elipse de $\mathcal{U}_1$ representa uma interação entre o usuário e o sistema. A interação inicial de $\mathcal{U}_1$, representada por uma seta sem origem, indica que o sistema apresenta um conjunto de elementos Seção, e nome é apresentado como item de dados (...Seção(nome)). O usuário pode, então, selecionar uma das seções apresentadas e o sistema retorna todas as matérias cadastradas para a seção selecionada. O resultado de cada interação que causa processamento do sistema deve ser representado por uma elipse separada, conectada à interação origem por uma seta. Assim, na segunda interação, o sistema retorna todas as matérias da seção selecionada, com os seguintes itens de dados: data de publicação, título e resumo. O mesmo raciocínio é seguido nas interações seguintes. As linhas finalizadas com um círculo, representadas na terceira interação, indicam as operações que podem ser realizadas no elemento Matéria.

Caso de Uso: *Ler matérias de uma seção*
Descrição:
1. O usuário seleciona uma das seções apresentadas pela aplicação
2. A aplicação retorna uma lista das matérias da seção selecionada, contendo a data de publicação, o título e o resumo de cada matéria
3. O usuário seleciona a matéria de interesse
4. A aplicação retorna o título, a data de publicação, o conteúdo da matéria, o nome do autor e uma lista das matérias relacionadas, caso exista, exibindo seus titulo e data de publicação. Se desejar, o usuário pode imprimir a matéria ou enviá-la para um amigo
5. O usuário solicita informações sobre o autor
6. A aplicação retorna o nome, a foto e a biografia do autor
Alternativa:
No item 5, o usuário pode solicitar ver as informações detalhadas de uma matéria relacionada. Neste caso, retorna para o item 4.

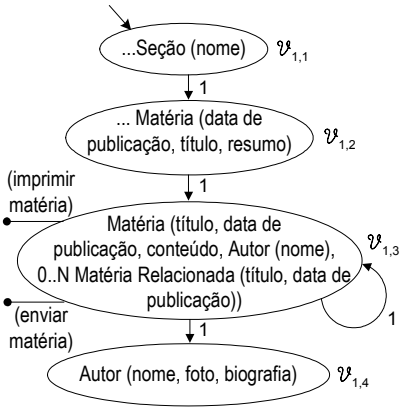


Figura 4.1: Tarefa *Ler matérias de uma seção* - Caso de uso e UID

Cada interação do UID possui associada uma VCN, a qual especifica os requisitos de conteúdo no contexto dessa interação, e as operações que se aplicam aos objetos da VCN. Por exemplo, os dados apresentados pelo sistema durante a interação inicial de u_1, ou seja, nome de todas as seções cadastradas, estão associados a uma visão sobre o banco de dados, a qual denominamos $v_{1,1}$. Assim, o UID u_1 mostrado na Figura 4.1 possui 4 VCN's ($v_{1,1}$, $v_{1,2}$, $v_{1,3}$ e $v_{1,4}$).

Em geral, as VCN's possuem uma estrutura da forma $E_P(e_1, e_2, ..., e_n)$, onde E_P é o elemento primário da VCN, que pode ser monovalorado ou multivalorado, e e_1, e_2, ..., e_n são sub-elementos de E_P. Cada e_i, por sua vez, pode ser um elemento simples ou complexo; neste último caso, e_i é dotado de estrutura própria. Algumas VCN's apresentam elementos que representam entrada de dados de usuários. Essas VCN's possuem um elemento primário, embora esse elemento não apareça explicitamente na interação. Considere, por exemplo, a VCN $v_{1,2}$ do UID u_1, que contém todas as matérias da seção selecionada em v_{11}. A estrutura de $v_{1,2}$ é Matéria(data de publicação, título, resumo). O elemento primário Matéria de v_{12} possui 3 sub-elementos simples: data de publicação, título e resumo. Já a VCN $v_{2,1}$ do UID u_2 (Figura 4.2) é composta de elementos que são entradas de dados do usuário, e sua estrutura é dada por Leitor(email, senha).

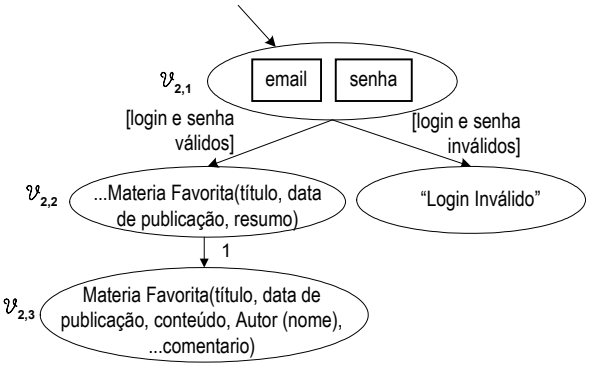


Figura 4.2: UID da tarefa "*Consultar matérias favoritas do leitor"*

4.2 Projeto das Visões de Objeto

O projeto das visões de objeto é realizado através da integração das VCN's dos UID's, de forma que essas visões satisfaçam os requisitos de todas as VCN's dos UID's. Como mostrado na Figura 4.3, cada VCN está relacionada com uma visão de objeto (visão de objeto base) sobre a qual a VCN é especificada. No caso de VCN's semelhantes, i.é., que têm objetos em comum, essas devem compartilhar a mesma visão de objeto. Nesse caso, o tipo da visão de objeto deve satisfazer os requisitos de conteúdo de todas as VCN's.

Cada visão de objeto está relacionada a uma ou mais tabelas do banco de dados (vide Figura 4.3). As visões de objeto são especificadas através de um conjunto de Assertivas de Correspondências (AC), que especificam as correspondências do esquema da visão de objeto com o esquema do banco de dados. As VCN's são especificadas através de um conjunto de Assertivas de Correspondências (AC) que especificam as correspondências da VCN com o esquema da visão de objeto. Como mostraremos no Capítulo 5, a vantagem do uso de AC's para especificação das VCN's e das visões de objetos consiste em permitir que a implementação e a manutenção das VCN's possam ser realizadas de forma automática, a partir de suas especificações conceituais.

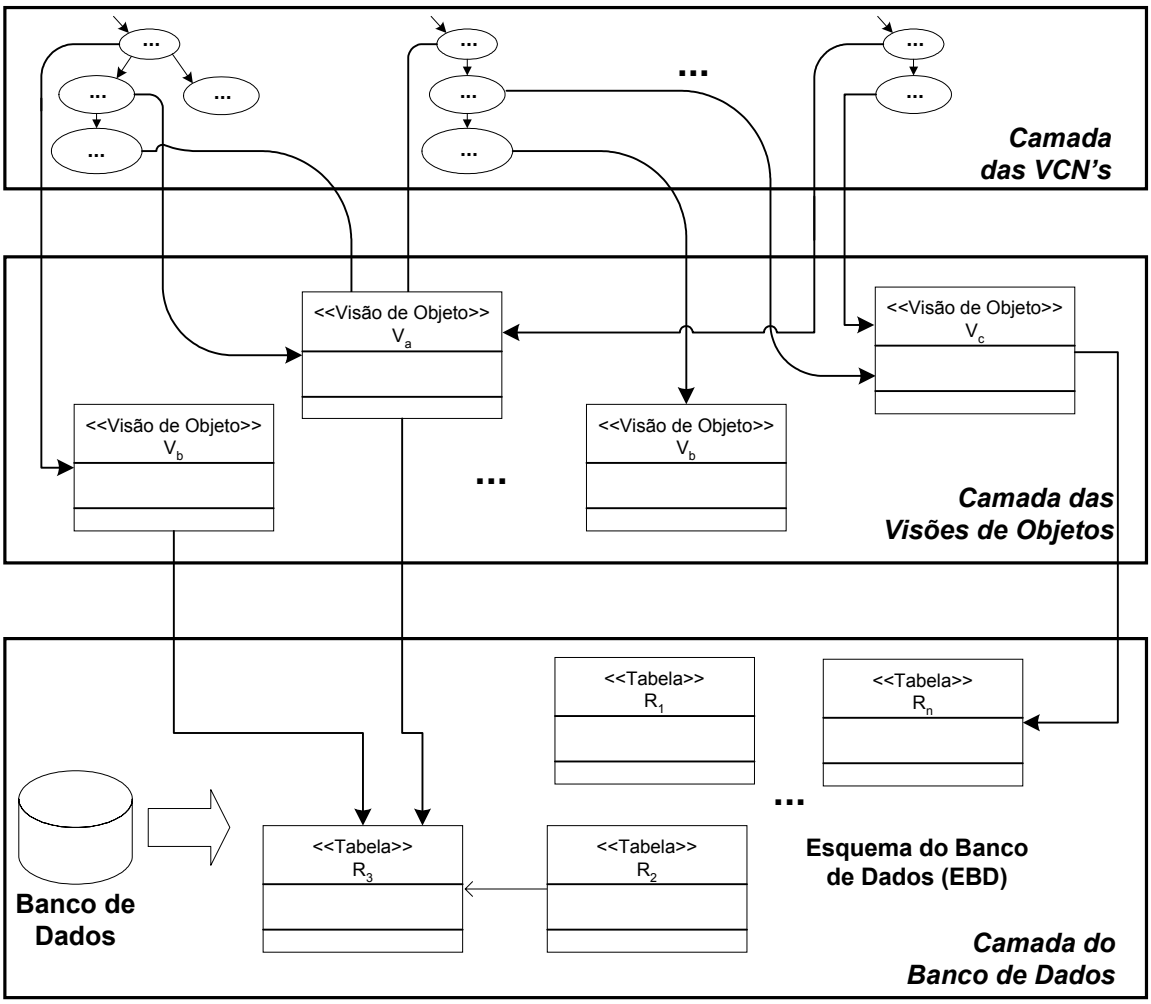


Figura 4.3: Projeto em 3 camadas

O projeto das visões de objeto compreende duas sub-atividades: Modelagem dos Tipos de Objetos das Visões e Especificação das Visões de Objetos. A seguir, descrevemos cada uma das sub-atividades.

4.2.1 Modelagem dos Tipos das Visões de Objeto

Cada visão de objeto possui um tipo de objeto associado, de modo que os objetos da visão são instâncias desse tipo. Dado que uma visão de objeto pode ser compartilhada por

várias VCN's, o tipo de objeto da visão deve satisfazer os requisitos de conteúdo de todas as VCN's.

A modelagem dos tipos de objetos das visões compreende 2 passos. Primeiro, a partir das estruturas das VCN's, são definidos os tipos de objetos das VCN's dos UID. Em seguida, os tipos de objeto das VCN's devem ser integrados de forma a eliminar redundâncias. A seguir, descrevemos cada um dos passos:

Passo 1: Geração dos tipos de objetos das VCN's

Neste passo, cada UID obtido na atividade Levantamento de Requisitos é analisado com o objetivo de definir os tipos das VCN's do UID. Os tipos das VCN's são definidos a partir das estruturas das VCN's do UID. Nesse trabalho, adaptamos as diretrizes propostas em [50], que definem como obter um diagrama de classes a partir de um dado UID. O processo de geração dos tipos de objetos compreende 2 sub-passos, descritos a seguir.

Passo 1.1: Primeiro, são definidos os tipos dos objetos das VCN's do UID, com seus atributos e operações. Usamos a notação gráfica apresentada na seção 2.2 para representação dos tipos de objetos das VCN's. Para cada VCN, é definido o tipo do seu elemento primário, conforme descrito a seguir:

- cada sub-elemento do elemento primário da VCN é mapeado em um atributo do tipo;
- o tipo do atributo é definido de acordo com a estrutura do sub-elemento; se o sub-elemento é estruturado, o tipo do atributo é definido a partir da estrutura desse elemento;
- a cardinalidade das inversas dos atributos de tipo estruturado deve ser definida, com base nas informações obtidas dos usuários;
- os tipos das VCN's com o mesmo elemento primário são integrados em um único tipo, de acordo com o padrão P2 apresentado no final desta seção;
- as operações do tipo da VCN são definidas com base nas operações aplicadas aos objetos da VCN;
- dentre os atributos do tipo gerado, é identificada a chave primária do tipo. Caso essa chave não apareça dentre os seus atributos, a mesma deve ser adicionada como atributo do tipo e definida como sua chave primária.

Considere, por exemplo, o UID $\mathcal{U}_1$, mostrado na Figura 4.4. Para os elementos primários Seção, Materia e Autor das VCN's de $\mathcal{U}_1$, foram gerados os tipos T_{secao}, $T_{materia}$, e T_{autor}, respectivamente. Note que o tipo de objeto $T_{materia}$ foi definido a partir das estruturas das VCN's $\mathcal{V}_{1,2}$ e $\mathcal{V}_{1,3}$, que possuem o mesmo elemento primário Matéria. De acordo com os requisitos dos usuários, na aplicação, cada matéria pode estar relacionada ao mesmo tempo com diversas matérias e um autor pode escrever diversas matérias. Assim, a cardinalidade de mat_rel^{-1} e $autor^{-1}$ é n.

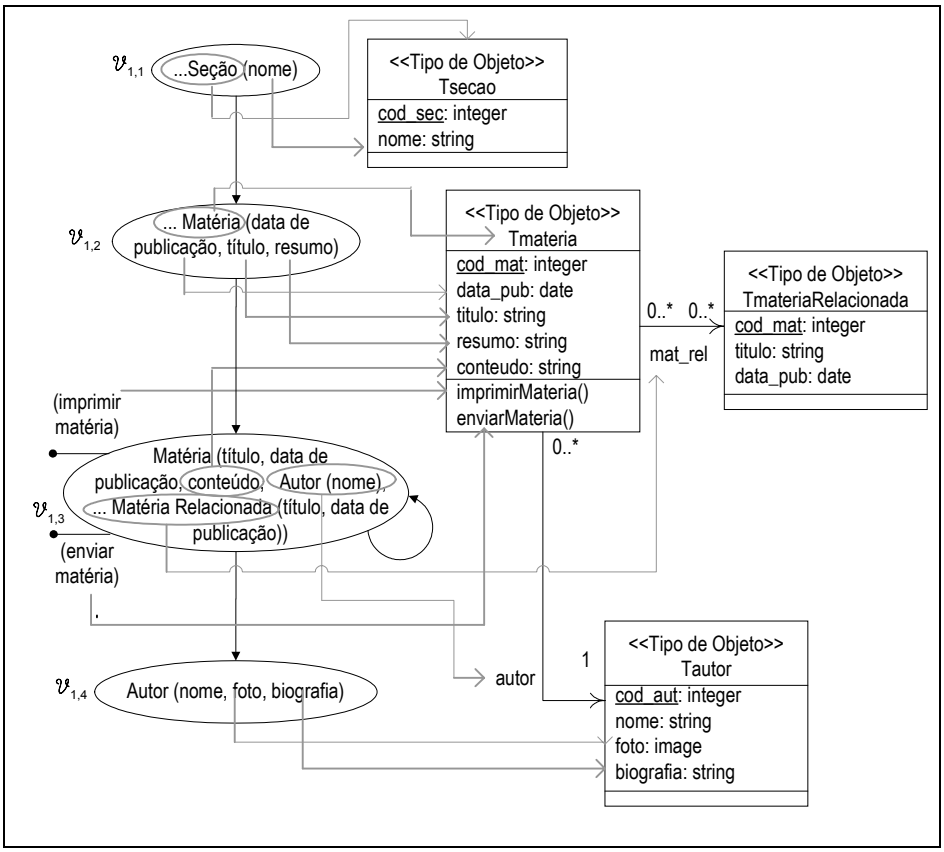


Figura 4.4: Tipos das VCN's do UID $\mathcal{U}_1$

Passo 1.2: Em seguida, deve-se verificar os casos de fluxo de informação (representado por uma seta) no UID que requerem a existência de um relacionamento entre objetos da interação origem com objetos da interação destino. Nestes casos, deve-se criar um atributo no tipo de objeto da VCN origem, caso esse ainda não exista. O tipo e a cardinalidade desse atributo são os mesmos do elemento primário da VCN destino.

Considere, por exemplo, o UID $\mathcal{U}_1$, mostrado na Figura 4.5. A partir do fluxo de informação entre a primeira e a segunda interação de $\mathcal{U}_1$, o atributo materias é adicionado ao tipo T_{secao}, que é o tipo da VCN $\mathcal{V}_{1,2}$. materias é definido como um atributo multivalorado e seu tipo é $T_{materia}$, pois esse é o tipo da VCN destino. De acordo com os requisitos dos usuários na aplicação, cada matéria pode pertencer a uma única seção e, assim, a cardinalidade de materias^{-1} é 1.

A Figura 4.6 mostra o diagrama de tipos das VCN's do UID $\mathcal{U}_2$, apresentado na Figura 4.2.

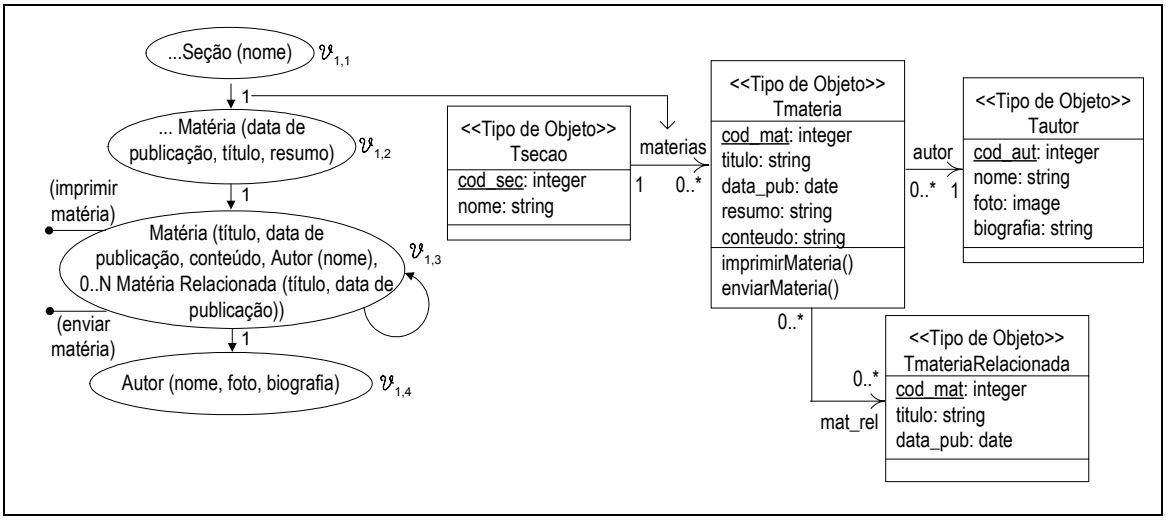


Figura 4.5: Atributo *matérias* identificado a partir de um fluxo de informação de $\mathcal{U}_1$

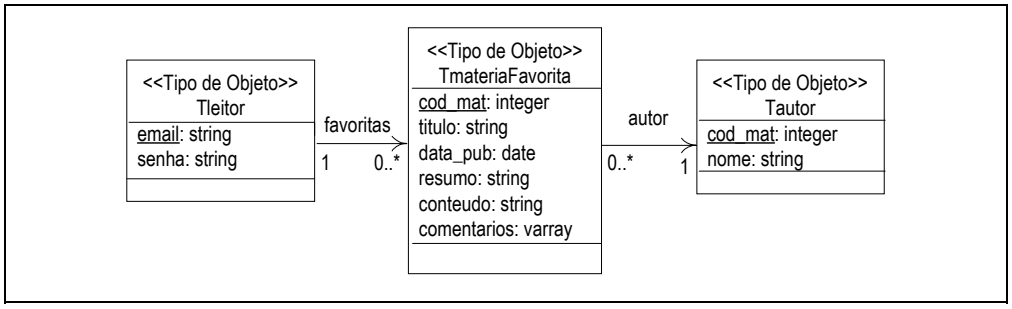


Figura 4.6: Tipos das VCN's do UID $\mathcal{U}_2$

Passo 2: Integração dos tipos

Nesse passo, os tipos de objeto obtidos no passo 1 devem ser integrados de forma a eliminar redundâncias. Para auxiliar no processo de integração dos tipos, propomos, a seguir, 3 padrões que asseguram que esse processo seja realizado de forma segura e eficiente: o padrão **P1**, "**Decompondo Tipos para Eliminar Dependências Funcionais Indesejáveis**", é utilizado para eliminar redundâncias causadas pelas dependências funcionais; o **P2**, "**Integrando Tipos Semelhantes**", é utilizado para integrar os tipos das VCN's semelhantes, i.é., que têm objetos em comum; e, por último, o **P3**, "**Adicionando Relacionamentos Ocultos**", é utilizado para identificar relacionamentos ocultos existentes entre tipos.

A seguir, apresentamos os padrões P1, P2 e P3 utilizados neste passo. Esses padrões são definidos utilizando o seguinte formato:

- nome do padrão: nome usado para identificar univocamente o padrão;
- contexto: um padrão soluciona um problema inserido em um contexto específico;
- problema: descreve o problema que o padrão soluciona;
- solução: propõe a solução para o problema descrito;
- exemplo: facilita o entendimento do padrão.

P1. Decompondo Tipos para Eliminar Dependências Funcionais Indesejáveis

Contexto: Suponha o esquema **S** da Figura 4.7. Seja **A** um tipo e **At**(A) o conjunto de atributos do tipo. Considere que $\{a, x_1, x_2, ..., x_n, y_1, y_2, ..., y_m\} \in$ **At**(A), tal que a dependência funcional $x_1, x_2, ..., x_n \rightarrow y_1, y_2, ..., y_m$[2] é válida em **S** e $\{x_1, x_2, ..., x_n\} \notin$ **Chaves**(A).

[2]A dependência funcional $x_1, x_2, ..., x_n \rightarrow y_1, y_2, ..., y_m$ é válida em **S** se e somente se para quaisquer instâncias a_1 e a_2, de A, se $a_1 \bullet x_i = a_2 \bullet x_i$, $1 \leq i \leq n$, então $a_1 \bullet y_j \equiv a_2 \bullet y_j$, $1 \leq j \leq m$.

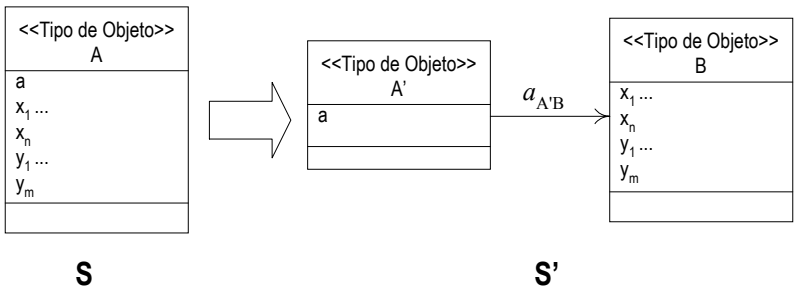


Figura 4.7: Padrão P1

Problema: Remover a redundância representada pela dependência funcional $x_1, x_2, ..., x_n \rightarrow y_1, y_2, ..., y_m$.

Solução: O esquema **S** da Figura 4.7 deve ser transformado no esquema **S',** como mostrado na Figura 4.7.

No esquema **S'**, o tipo **A** foi decomposto nos tipos **A'** e **B** e o atributo $a_{A'B}$ é definido no tipo **A'**, onde o tipo de $a_{A'B}$ é **B**. Os atributos de **S'** são distribuídos como se segue:

(i) $\mathbf{At}(A') = \mathbf{At}(A) - \{x_1, x_2, ..., x_n, y_1, y_2, ..., y_n\} + a_{A'B}$
(ii) $\mathbf{At}(B) = \{x_1, x_2, ..., x_n, y_1, y_2, ..., y_n\}$

Exemplo: Considere o esquema $\mathbf{S_1}$, mostrado na Figura 4.8. Analisando a semântica dos atributos do tipo $\mathbf{T_{materia}}$, verificamos que as seguintes dependências funcionais são válidas para $\mathbf{S_1}$.

$\mathbf{FD_1}$: cod_mat $\rightarrow$ titulo, resumo, conteudo, cod_autor
$\mathbf{FD_2}$: cod_autor $\rightarrow$ nome_autor

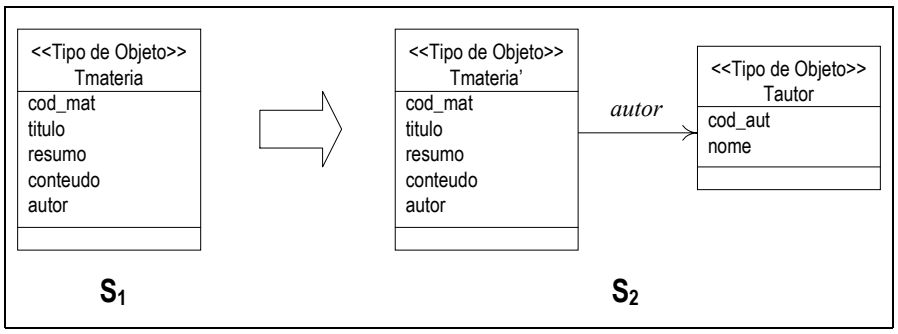


Figura 4.8: Exemplo do uso do padrão P1

A dependência funcional FD_1 especifica que o atributo cod_mat determina unicamente os atributos titulo, resumo, conteudo e cod_autor. A dependência funcional FD_2 especifica que o atributo cod_autor determina unicamente o atributo nome_autor. Como o atributo cod_autor não faz parte da chave de $T_{materia}$, FD_2 está indicando que o atributo nome, na verdade, é um atributo de um outro tipo que está embutido no tipo $T_{materia}$, cuja chave é o atributo cod_autor. Isso implica que o tipo $T_{materia}$ está representando mais de um tipo de objeto do mundo real. Nesse caso, propõe-se reestruturar o esquema S_1, como mostrado no esquema S_2, da Figura 4.8. No esquema S_2, o tipo $T_{materia}$ foi decomposto nos tipos $T_{materia}$' e T_{autor}. Os atributos cod_aut e nome_autor foram transferidos para o novo tipo T_{autor}, e o atributo autor foi definido no tipo $T_{materia}$', onde o tipo de autor é T_{autor}.

P2. Integrando Tipos Semelhantes

Contexto: A extensão de uma VCN é o conjunto de objetos membros da VCN em um determinado instante. Nesse padrão, os tipos das VCN's semelhantes, cujas extensões têm objetos em comum, devem ser identificados e integrados. Consideramos três tipos de correspondência semântica de extensões das VCN's: equivalência, subconjunto e sobreposição.

Problema: Identificar e integrar tipos das VCN's semelhantes, de forma a eliminar redundâncias existentes.

Solução: A semântica da aplicação deve ser analisada para identificar as VCN's semelhantes e o tipo de correspondência existente. As VCN's cujos nomes dos elementos primários são similares (sinônimos) indicam a possibilidade de serem semelhantes. Suponha a seguir $\mathbf{T_{V1}}$ e $\mathbf{T_{V2}}$ tipos das VCN's $\mathbf{V_1}$ e $\mathbf{V_2}$. Nos casos a seguir, consideramos três tipos de correspondência semântica entre $\mathbf{V_1}$ e $\mathbf{V_2}$: (i) A extensão de $\mathbf{V_1}$ é equivalente à extensão de $\mathbf{V_2}$ ($\mathbf{V_1 \equiv V_2}$), i.e., contém o mesmo conjunto de objetos; (ii) A extensão de $\mathbf{V_1}$ é um subconjunto da extensão de $\mathbf{V_2}$ ($\mathbf{V_1 \subseteq V_2}$); (iii) As extensões de $\mathbf{V_1}$ e $\mathbf{V_2}$ se sobrepõem ($\mathbf{V_1 \cap V_2 \neq \varnothing}$).

- Caso 1: $\mathbf{V_1 \equiv V_2}$

Nesse caso, os tipos $\mathbf{T_{V1}}$ e $\mathbf{T_{V2}}$ são integrados em um único tipo $\mathbf{T_{V12}}$, como mostrado na Figura 4.9. Seja $\mathbf{At}(T_{V1})$ o conjunto de atributos de $\mathbf{T_{V1}}$ e $\mathbf{At}(T_{V2})$ o conjunto de atributos de $\mathbf{T_{V2}}$, os atributos de S' são definidos como se segue:

(i) $\mathbf{At}(T_{V12}) = \mathbf{At}(T_{V1}) \cup \mathbf{At}(T_{V2})$, onde $\cup$ (união de conjuntos) integra todos os atributos sinônimos existentes entre os tipos T_{V1} e T_{V2}.

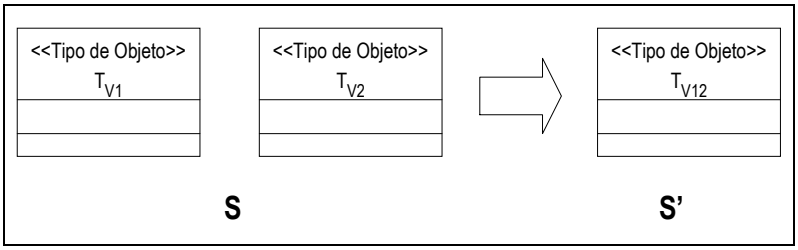


Figura 4.9: Padrão P2 (caso 1)

Exemplo: Considere o esquema $\mathbf{S_1}$ mostrado na Figura 4.10. Podemos identificar nesse esquema os tipos $\mathbf{T_{materia}}$ e $\mathbf{T_{publicacao}}$ com nomes sinônimos. Para analisar a semântica do mundo real, verificamos se toda matéria é uma publicação e vice-versa. Se for verdade, existe um mapeamento um para um entre as instâncias dos tipos $\mathbf{T_{materia}}$ e $\mathbf{T_{publicacao}}$. É necessário, portanto, integrar $\mathbf{T_{materia}}$ e $\mathbf{T_{publicacao}}$ em um novo tipo $\mathbf{T_{publicacao}}$, como mostrado no esquema $\mathbf{S_2}$ da Figura 4.10.

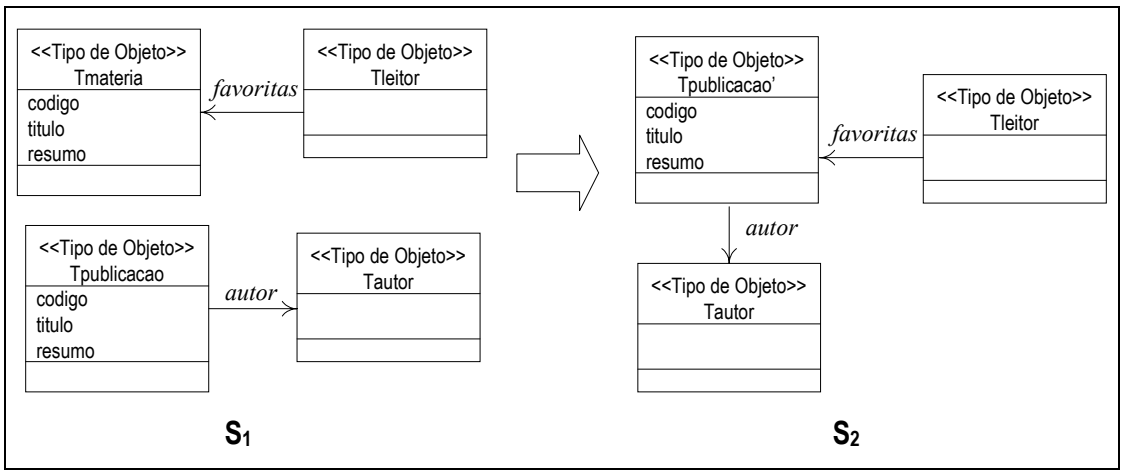


Figura 4.10: Exemplo do uso do padrão P2 (caso 1)

- Caso 2: $\mathbf{V_1 \subseteq V_2}$

Nesse caso, o esquema S deve ser transformado no esquema S', mostrado na Figura 4.11. Seja $\mathbf{At}(T_{V1})$ o conjunto de atributos de T_{V1} e $\mathbf{At}(T_{V2})$ o conjunto de atributos de T_{V2}, os atributos de S' são definidos como se segue:

(i) $\mathbf{At}(T_{V2}) = \mathbf{At}(T_{V2}')$

(ii) $\mathbf{At}(T_{V1}') = \mathbf{At}(T_{V1}) - \mathbf{At}(T_{V2})$, onde - (diferença de conjuntos) significa que os atributos do tipo T_{V1}' são aqueles atributos de T_{V1} que não possuem atributos semanticamente equivalentes em T_{V2}. T_{V1}' é definido como um subtipo de T_{V2}' e, portanto, herda todos os atributos desse tipo.

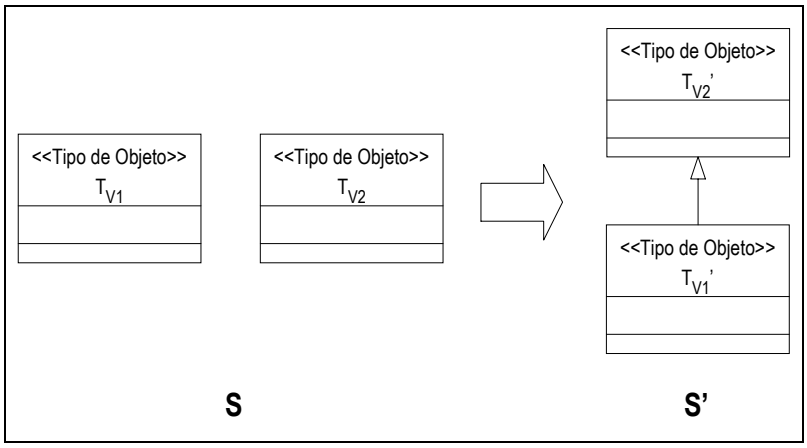


Figura 4.11: Padrão P2 (caso 2)

Exemplo: Considere o esquema $\mathbf{S_1}$ mostrado na Figura 4.12. Podemos identificar nesse esquema os tipos $\mathbf{T}_{publicacao}$ e $\mathbf{T}_{livro}$ com um conjunto de atributos comuns: codigo, titulo e resumo. Após analisar a semântica do mundo real, verificamos que a extensão de $\mathbf{T}_{livro}$ é um subconjunto da extensão de $\mathbf{T}_{publicacao}$. É necessário, portanto, reestruturar o esquema $\mathbf{S_1}$, obtendo o esquema $\mathbf{S_2}$, como mostrado da Figura 4.12.

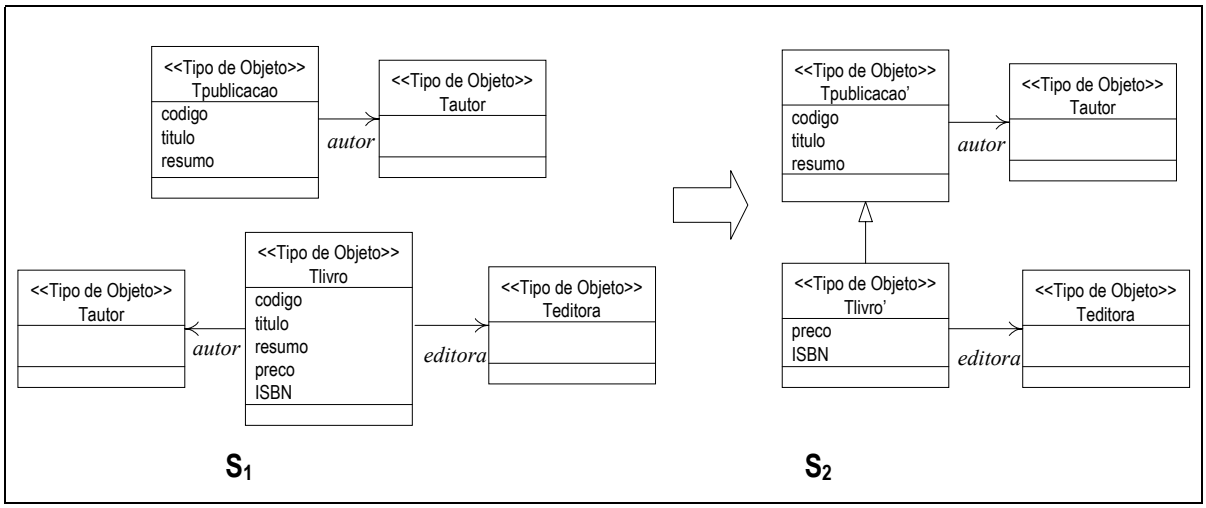


Figura 4.12: Exemplo do uso do padrão P2 (caso 2)

- Caso 3: $\mathbf{V_1} \cap \mathbf{V_2} \neq \varnothing$

Nesse caso, o esquema S deve ser transformado no esquema S', mostrado na Figura 4.13. Seja $\mathbf{At}(T_{V1})$ o conjunto de atributos de T_{V1} e $\mathbf{At}(T_{V2})$ o conjunto de atributos de T_{V2}, os **atributos** de S' são definidos como se segue:

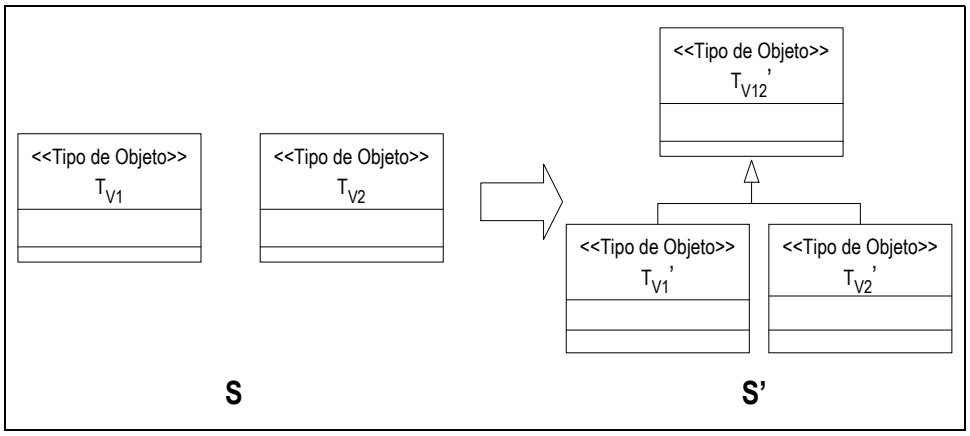


Figura 4.13: Padrão P2 (caso 3)

(i) **At**(T_{V1}') = **At**(T_{V1}') - **At**(T_{V2}'), onde - (diferença de conjuntos) significa que os atributos do tipo T_{V1}' são aqueles atributos de T_{V1} que não possuem atributos semanticamente equivalentes em T_{V2}.
(ii) **At**(T_{V2}') = **At**(T_{V2}) - **At**(T_{V1}), onde - (diferença de conjuntos) significa que os atributos do tipo T_{V2}' são aqueles atributos de T_{V2} que não possuem atributos semanticamente equivalentes em T_{V1}.
(iii) **At**(T_{V12}') = **At**(T_{V1}) ∩ **At**(T_{V2}), onde ∩ (interseção de conjuntos) significa que os atributos do tipo T_{V12}' são aqueles atributos de T_{V1} que possuem atributos semanticamente equivalentes em T_{V2}.

Exemplo: Considere o esquema $\mathbf{S_1}$ mostrado na Figura 4.12. Podemos identificar nesse esquema os tipos $\mathbf{T}_{autor}$ e $\mathbf{T}_{leitor}$ com os atributos cod_pes, nome e email em comum. Após analisar a semântica do mundo real, verificamos que $\mathbf{T}_{livro} \cap \mathbf{T}_{publicacao} \neq \varnothing$. É necessário reestruturar o esquema $\mathbf{S_1}$, obtendo o esquema $\mathbf{S_2}$, mostrado da Figura 4.14.

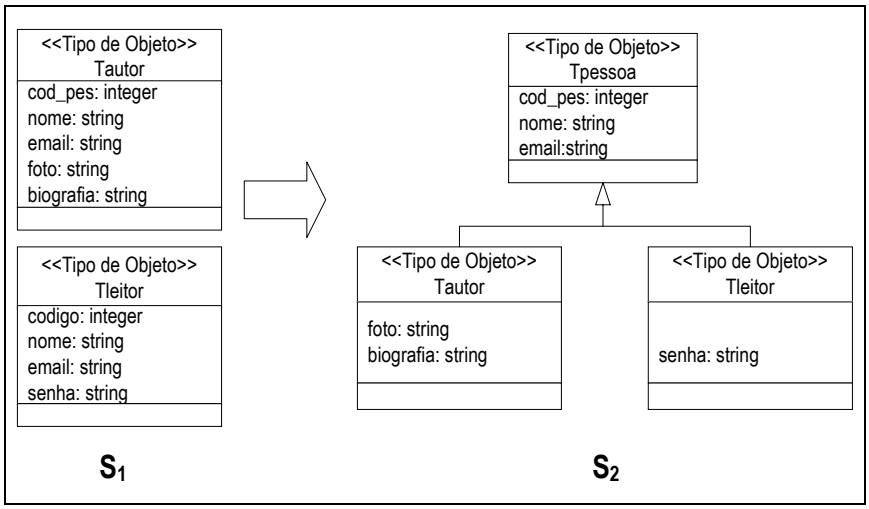


Figura 4.14: Exemplo do uso do padrão P2 (caso 3)

P3. Adicionando Relacionamentos Ocultos

Contexto: Algumas situações de modelagem sugerem a possibilidade de existir uma restrição de dependência existencial entre os tipos. Suponha o esquema **S** da Figura 4.15. Sejam **A** e **B** tipos, **At**(A) o conjunto de atributos de A, **At**(B) o conjunto de atributos de B e

$\mathbf{K}_2$ a chave do tipo B. As situações que sugerem a possibilidade de existir uma restrição de dependência existencial[3] entre os tipos A e B são:

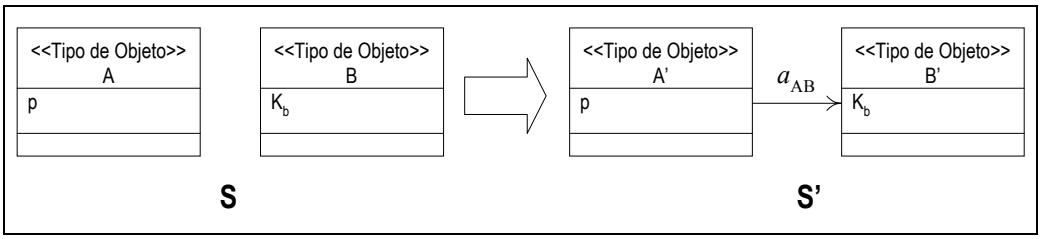


Figura 4.15: Padrão P3

1. Existe um atributo $\mathbf{p} \in \mathbf{At}(A)$ com nome similar ao nome do tipo B.
2. Existe um atributo $\mathbf{p} \in \mathbf{At}(A)$ com nome e tipo similares ao nome e tipo do atributo chave $\mathbf{K}_b$ da classe B.

Problema: Verificar se existe uma restrição de dependência existencial entre os tipos A e B envolvendo os atributos **p** e $\mathbf{K}_b$. Caso exista, o esquema conceitual deve ser reestruturado para capturar essa restrição.

Solução: A semântica da aplicação deve ser verificada. De acordo com a análise da semântica, temos que existe uma restrição entre os tipos A e B, dada por $A[p] \subset B[K_b]$, se para toda instância a de A existe uma instância b de B tal que $a \bullet p = b \bullet K_b$. Se existe uma restrição entre os tipos A e B, dada por $A[p] \subset B[K_b]$, o esquema **S** deve ser transformado no esquema **S'**, como mostrado na Figura 4.15.

No esquema **S'**, o atributo $a_{A'B}$ é definido no tipo **A'**, onde o tipo de $a_{A'B}$ é **B'**. Os atributos de **S'** são distribuídos como se segue:

(i) $\mathbf{At}(A') = \mathbf{At}(A) - \{\mathbf{p}\} + \{a_{AB}\}$
(ii) $\mathbf{At}(B') = \mathbf{At}(B)$

[3] Sejam $a_1, a_2, ..., a_n$ atributos de A e $b_1, b_2, ..., b_n$, atributos de B. A restrição de dependência existencial de subconjunto $A[a_1, a_2, ..., a_n] \subset B[b_1, b_2, ..., b_n]$ especifica que para qualquer instância a de A, existe uma instância b de B tal que $a.a_i = b.b_i$, $1 \leq i \leq n$.

Exemplo: Considere o esquema $\mathbf{S_1}$ mostrado na Figura 4.16. Pela semântica do mundo real, temos que, nesse esquema, existe uma restrição entre os tipos $\mathbf{T}_{materia}$ e $\mathbf{T}_{autor}$, que especifica que para toda instância m de $\mathbf{T}_{materia}$, existe uma instância a de $\mathbf{T}_{autor}$, tal que m.autor = a.cod_aut. O atributo autor, em vez de representar um atributo do tipo $\mathbf{T}_{materia}$, na verdade representa um relacionamento oculto entre os tipos $\mathbf{T}_{materia}$ e $\mathbf{T}_{autor}$.

Para capturar a restrição de dependência existencial identificada entre os tipos $\mathbf{T}_{materia}$ e $\mathbf{T}_{autor}$, deve-se reestruturar o esquema $\mathbf{S_1}$, como mostrado no esquema $\mathbf{S_2}$ da figura. No esquema $\mathbf{S_2}$, criamos dois novos tipos $\mathbf{T}_{materia}$' e $\mathbf{T}_{autor}$' e definimos o atributo de ligação autor entre eles.

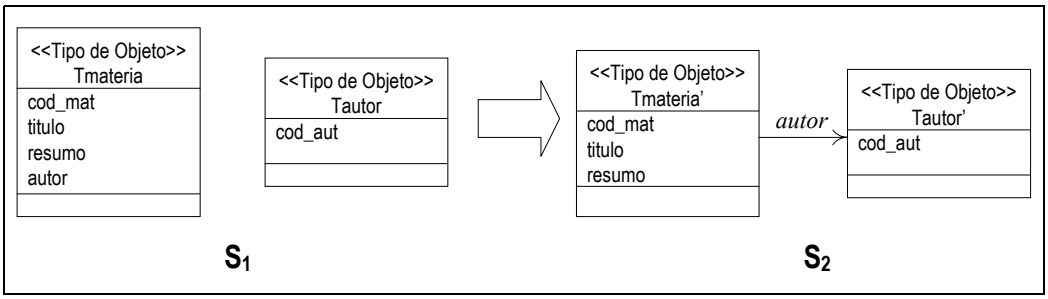


Figura 4.16: Exemplo do uso do padrão P3

A Figura 4.17 mostra o diagrama de tipos obtido da integração dos tipos das VCN's dos UID $\mathcal{U}_1$ e $\mathcal{U}_2$, mostrados nas Figura 4.5 e Figura 4.6, respectivamente. Primeiro, o padrão P2 foi utilizado para integrar os tipos semelhantes $\mathbf{T}_{materiaRelacionada}$ e $\mathbf{T}_{materia}$, ambos do diagrama de tipos do UID $\mathcal{U}_1$, e os tipos $\mathbf{T}_{autor}$ dos dois diagramas. O padrão P1 foi utilizado para identificar a dependência funcional *cod_mat* → *titulo*, *data_pub*, *resumo*, *conteudo* do tipo $T_{materiaFavorita}$ e para definir um novo tipo, de forma a remover essa dependência de $T_{materiaFavorita}$. O tipo criado é então integrado com o tipo $\mathbf{T}_{materia}$ do diagrama de tipos do UID $\mathcal{U}_1$.

No estudo de caso apresentado no Apêndice A, analisamos 3 casos de uso para a aplicação Publicações.

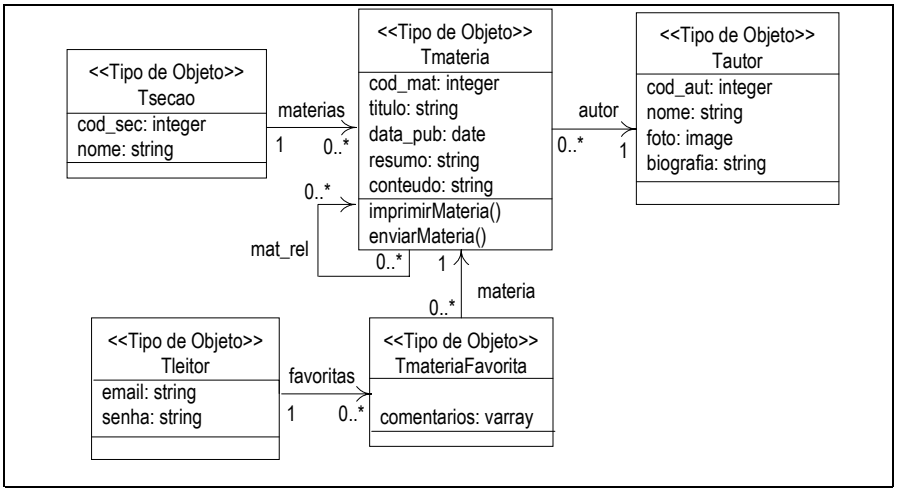


Figura 4.17: Diagrama de tipos para a aplicação Publicações

4.2.2 Especificação das Visões de Objeto

No passo anterior (modelagem dos tipos das visões de objeto), foram gerados os tipos de objetos das VCN's, de forma que cada VCN terá associado um tipo de objeto, o qual chamamos de tipo base da VCN. Neste passo, serão especificadas as visões de objeto para os tipos bases das VCN's. No nosso enfoque, a especificação de uma visão de objeto é uma tripla $\mathbf{V=< R_b, T_v\ A_v>}$, onde $\mathbf{R_b}$ é a tabela base, $\mathbf{T_v}$ é o tipo dos objetos da visão, e $\mathbf{A_v}$ é um conjunto de assertivas de correspondências que especificam os relacionamentos dos atributos do tipo $\mathbf{T_v}$ com os atributos/caminhos do tipo da tabela base.

Como mostrado na Figura 4.3, as VCN's são especificadas sobre a visão de objeto do seu tipo base (visão de objeto base), e as visões de objetos são especificadas sobre o esquema do banco. A Figura 4.18 mostra o diagrama de tipos da aplicação Publicações da Figura 4.17, com as respectivas visões de objetos. Note que os atributos cujo tipo possui uma visão de objeto associada são redefinidos como atributos de referência ao tipo.

Neste trabalho, usamos cartões de especificação de visão para visualizar a especificação de uma visão de objeto. A Figura 4.20 mostra o cartão de especificação da visão de objeto Materias_v da aplicação Publicações (Figura 4.18), definida sobre o esquema do banco de dados relacional Publicacoes_rel mostrado na Figura 4.19.

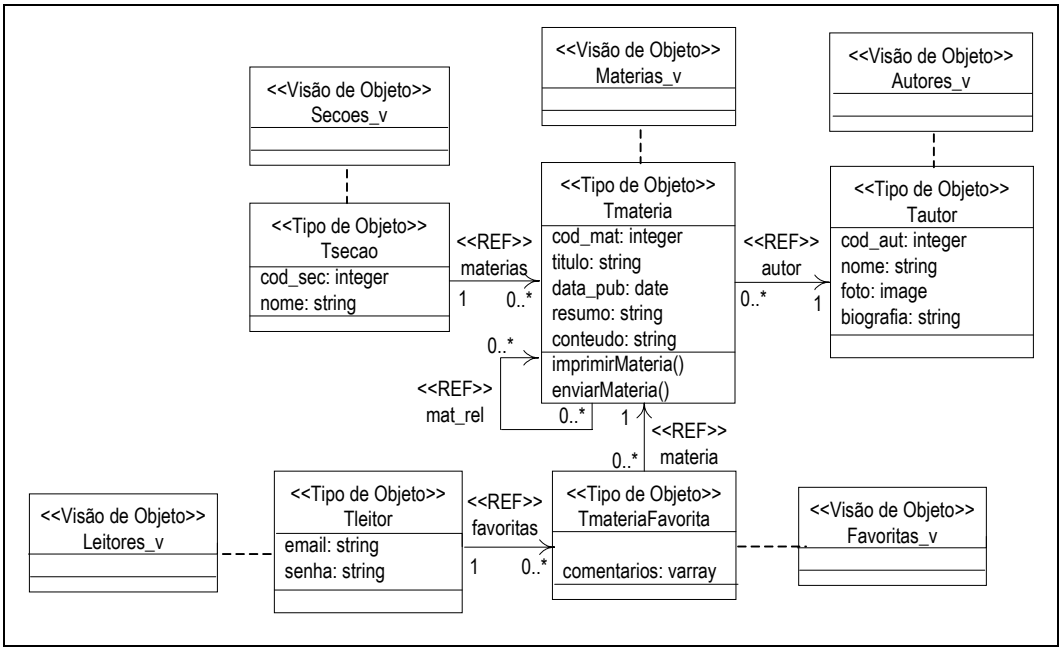


Figura 4.18: Visões de Objetos da aplicação Publicações

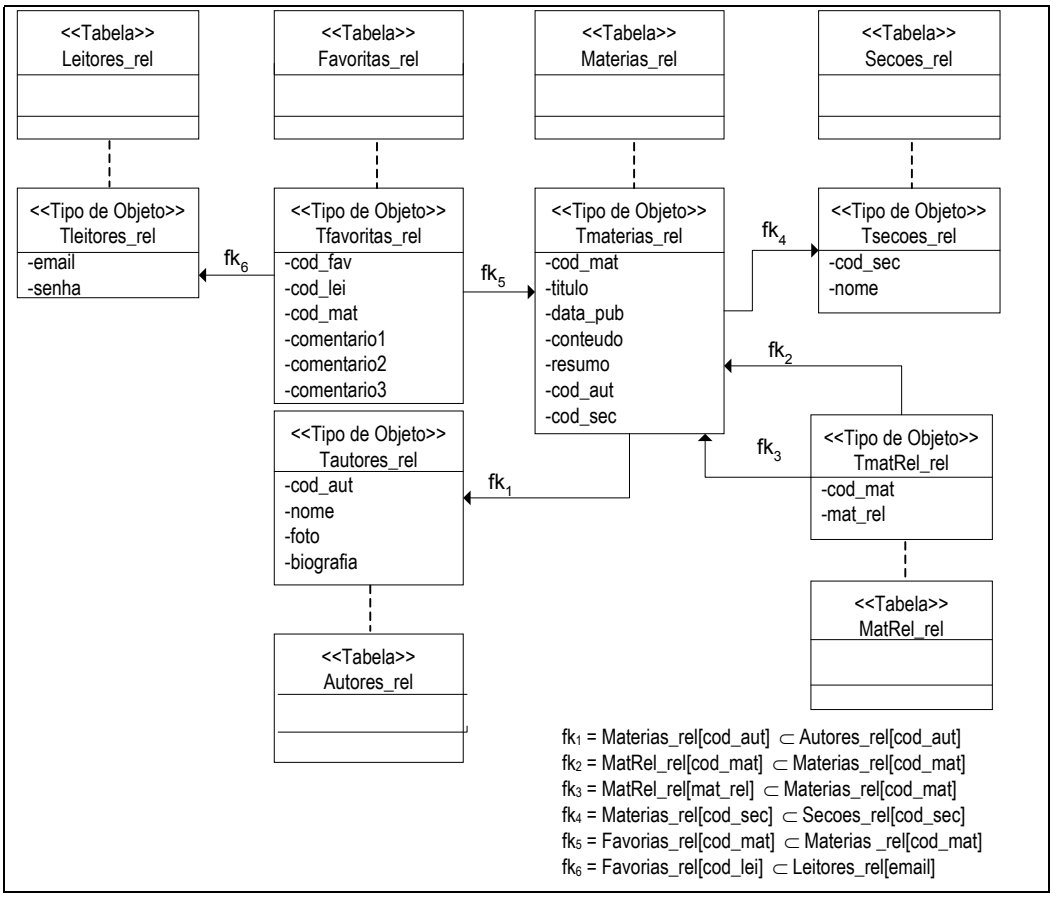


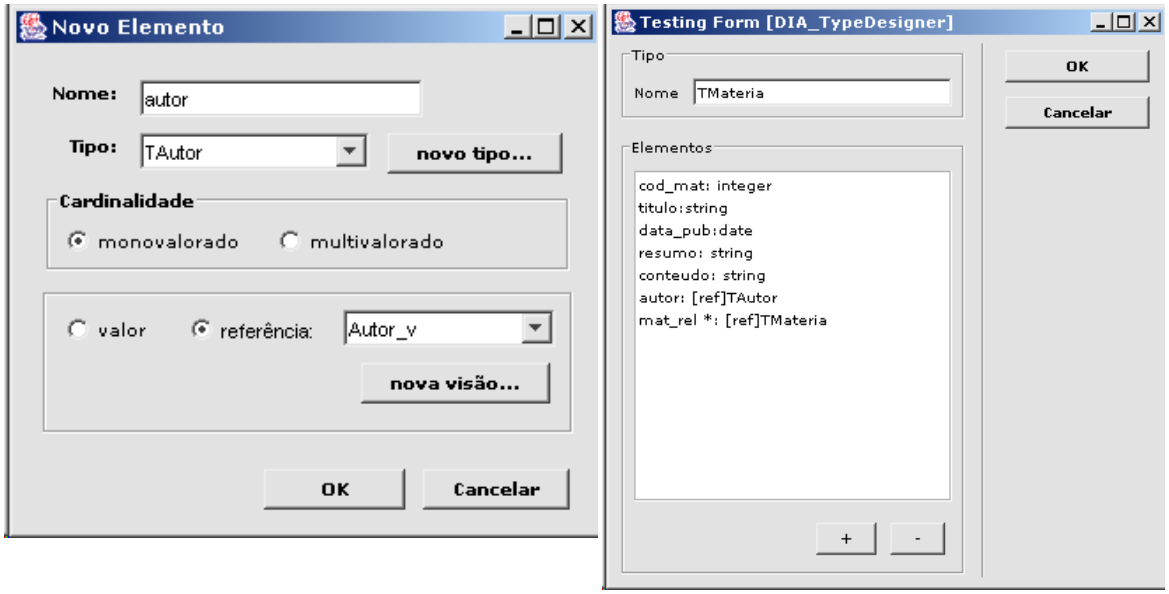
Figura 4.19: Esquema Relacional do Banco de Dados Publicações_rel.

Visão de Objeto: Materias_v	
Tabela Base:	(**ACE**: ψ_1: [Materias_v] ≡ [Materias_rel])
Materias_rel	(**ACO:** ψ_2: [$T_{materia}$, {cod_mat}] ≡ [$T_{materias\_rel}$, {cod_mat}])
Atributos:	**ACC's:**
cod_mat: integer	ψ_3: [$T_{materia}$•cod_mat] ≡ [$T_{materias\_rel}$•cod_mat]
data_publicacao: string	ψ_4: [$T_{materia}$•data_publicação] ≡ [$T_{materias\_rel}$•data_pub]
titulo: string	ψ_5: [$T_{materia}$•titulo] ≡ [$T_{materias\_rel}$•titulo]
resumo: string	ψ_6: [$T_{materia}$•resumo] ≡ [$T_{materias\_rel}$•resumo]
conteudo: string	ψ_7: [$T_{materia}$•conteudo] ≡ [$T_{materias\_rel}$•conteudo]
autor: REF <<T_{autor}>>	ψ_8: [$T_{materia}$•autor] ≡ [$T_{materias\_rel}$•fk_1]
mat_rel: set of REF <<$T_{materia}$>>	ψ_9: [$T_{materia}$•mat_rel] ≡ [$T_{materias\_rel}$•fk_2^{-1}•fk_3]

Figura 4.20: Cartão de especificação da visão de objeto Materias_v

As especificações das visões de objeto podem ser realizadas de forma gráfica com a ferramenta para Geração da Especificação de Visão de Objeto, proposta em [42]. A seguir, descrevemos a geração da especificação da visão de objeto Materias_v (Figura 4.18) com a ferramenta. O usuário começa definindo, a partir de uma interface gráfica da ferramenta, o nome da visão e o nome do seu tipo. Em seguida, o usuário deve criar os atributos do tipo da visão (Figura 4.21 (a)). Para isso, usa a tela de criação de atributo (Figura 4.21 (b)). Além do nome, são pedidos o tipo do atributo, sua cardinalidade e se o mesmo é de valor ou de referência. No caso de atributos de tipo estruturado, então um novo tipo deve ser criado (nome e atributos). No caso de atributos de referência, deve-se definir seu escopo (visão de objeto referenciada). Se a visão ainda não existe, deverá ser criada.

Em seguida, o usuário seleciona, de uma lista de tabelas e visões do banco de dados, a tabela base ou visão base, e, se necessário, define a condição de seleção. Para a visão Materias_v**,** a tabela Materias_rel é selecionada e a ACE $\psi_1^{[EC]}$:[Materias_v] ≡ [Materias_rel] é gerada.



(a) (b)

Figura 4.21: Telas da ferramenta: (a)Editor de Tipo; (b)Editor de Atributo.

O usuário deve então realizar o *matching* [5][38][29] do tipo da visão com o tipo da tabela base. Para isso, a ferramenta exibe uma tela contendo, em formato de árvore de diretório, a estrutura do tipo da visão e a estrutura do tipo base (Figura 4.22). Observe que para o tipo base, além dos atributos, são mostradas as demais ligações do tipo (ligações de chave estrangeira e inversas), de modo que o usuário possa definir caminhos que naveguem por essas ligações. Para definir a ACC de um atributo da visão, o usuário relaciona graficamente o atributo da visão com um atributo ou caminho do tipo da tabela base. A seguir mostramos como gerar as ACC's para alguns dos atributos do tipo $T_{materia}$:

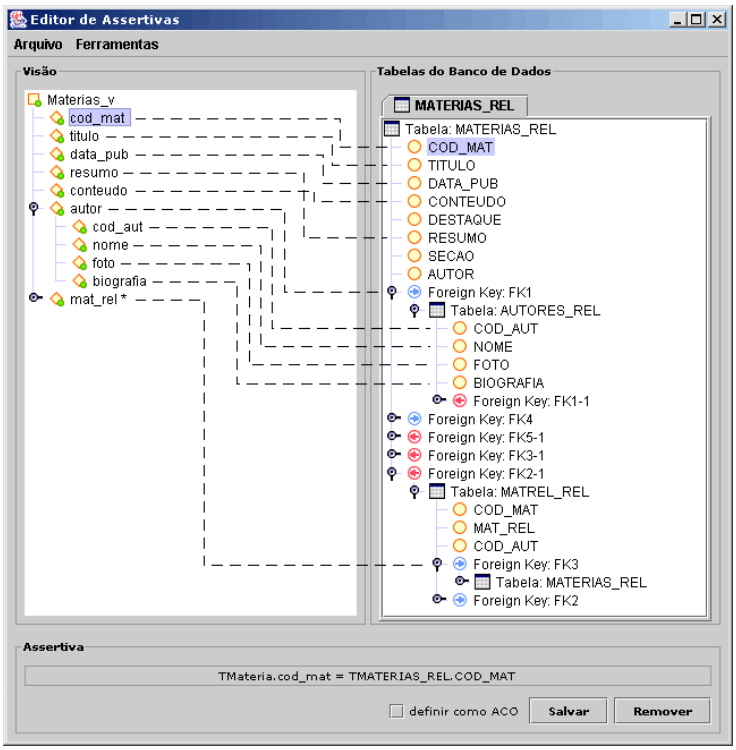


Figura 4.22: Tela da ferramenta para editar assertivas de correspondência.

- Para definir a ACC do atributo cod_mat, o usuário seleciona cod_mat no esquema da visão e cod_mat no esquema do banco de dados. Ao clicar no botão "Salvar", a ferramenta mostra a ACC gerada ($\psi_3^{[EC]}$: $\mathbf{T}_{materia}\bullet cod\_mat] \equiv [T_{materias\_rel}\bullet cod\_mat]$);
- Para definir a ACC do atributo autor, o usuário seleciona autor no esquema da visão e fk_1 no esquema do banco. Como autor é um objeto de tipo T_{autor}, que é um tipo estruturado, o usuário deve então definir recursivamente as correspondências para os atributos de T_{autor}.
- Para definir a ACC do atributo mat_rel, o usuário seleciona mat_rel no esquema da visão e $fk_2^{-1}\bullet fk_3$ no esquema do banco. Observe que, para o atributo mat_rel, o "*" denota que o atributo é multivalorado e $T_{materia}$, especificado entre parênteses, é o tipo dos objetos em mat_rel. Como $T_{materia}$ é um tipo estruturado, o usuário deve então definir recursivamente as correspondências para os atributos de $T_{materia}$.

O cartão de especificação de Materias_v (vide Figura 4.20) é gerado automaticamente pela ferramenta. Como veremos no Capítulo 5, as visões de objeto podem ser implementadas de forma automática a partir de suas especificações.

4.3 Projeto das Visões de Contexto de Navegação

Durante o Projeto das VCN's, cada VCN dos UID's é especificada como uma visão de seleção sobre uma visão de objeto. Para a especificação conceitual das VCN's, usamos *cartões de especificação*, que são uma adaptação dos cartões de especificação de contexto propostos em Rossi [40]. Na Figura 4.23, é mostrado o cartão de especificação da VCN $\mathcal{V}_{1,3}$ do UID $\mathcal{U}_1$ (Figura 4.1), a qual é especificada sobre a visão de objeto Materias_v (Figura 4.18).

VCN: $\mathcal{V}_{1,3}$	
Parâmetros: p : integer	
Visão de Objeto Base:	**Condição de Seleção** (**ACE**: $\psi_1^{[V1,3]}$: $V_{1,3} \equiv$ Materias_v[cod_mat = p])
Materias_v	cod_mat = p
Atributos:	**Assertivas de Correspondência**
titulo: string	$\psi_2^{[V1,3]}$: $T_{[V1,3]}$•titulo ≡ Tmateria•titulo
data_publicação: string	$\psi_3^{[V1,3]}$: $T_{[V1,3]}$•data_publicação ≡ Tmateria•data_pub
conteudo: string	$\psi_4^{[V1,3]}$: $T_{[V1,3]}$•conteudo ≡ Tmateria•conteudo
↘ autor: <$T_{[autor]}$>	$\psi_5^{[V1,3]}$: $T_{[V1,3]}$•autor ≡ Tmateria•autor
cod_aut: integer	$\psi_6^{[V1,3]}$: $T_{[autor]}$•cod_aut ≡ Tautor•cod_aut
nome: string	$\psi_7^{[V1,3]}$: $T_{[autor]}$•nome ≡ Tautor•nome
↘ mat_rel: set of <$T_{[mat\_rel]}$>	$\psi_8^{[V1,3]}$: $T_{[V1,3]}$•mat_rel ≡ Tmateria•mat_rel
cod_mat: integer	$\psi_9^{[V1,3]}$: $T_{[mat\_rel]}$•cod_mat ≡ Tmateria•cod_mat
titulo: string	$\psi_{10}^{[V1,3]}$: $T_{[mat\_rel]}$•titulo ≡ Tmateria•titulo
data_publicação: string	$\psi_{11}^{[V1,3]}$: $T_{[mat\_rel]}$•data_publicação ≡ Tmateria•data_pub
Elos:	
inf_autor: {**Destino**: V_{14} ; **Parâmetros**:{self.autor.cod_aut}}	
inf_mat_rel:{**Destino**:V_{13}; **Parâmetros**:{self.mat.cod_mat | mat ∈ mat_rel}}	
Operações:	
imprimirMateria()	
enviarMateria()	
Ordenação:	

Figura 4.23: Cartão de Especificação da VCN $\mathcal{V}_{1,3}$

Os campos de um cartão de especificação são descritos a seguir:

- Visão de objeto base: especifica a visão de objeto base da VCN. Para cada VCN do UID, existe uma visão de objeto no esquema conceitual, denominada *visão de objeto base*, da qual são selecionados os objetos da VCN. Várias VCN's podem compartilhar a mesma visão de objeto base; nesse caso, o tipo da visão de objeto satisfaz os requisitos de conteúdo de todas as VCN's. Por exemplo, as VCN's $\mathcal{V}_{1,2}$ e $\mathcal{V}_{1,3}$ possuem a mesma visão de objeto

base **Materias_v**, e todos os atributos de $\mathcal{V}_{1,2}$ e $\mathcal{V}_{1,3}$ podem ser derivados a partir dos atributos do tipo base $\mathbf{T}_{materia}$.

- *Parâmetros:* especifica os parâmetros usados na ***condição de seleção***, que define a condição que os objetos da VCN devem satisfazer. Assim, temos que a VCN $\mathcal{V}_{1,3}$ é uma visão de seleção que contém todos os objetos de Materias_v, cujos valores do atributo titulo são iguais ao parâmetro p. É importante notar que, definidas a *visão de objeto base* e a *condição de seleção*, pode-se inferir a AC de extensão da VCN. A ACE de $\mathcal{V}_{1,3}$ é dada por $\psi_1^{[V1,3]}$: **[$V_{1,3}$] ≡ [Materias_v[titulo=p]]**.

- *Propriedades:* define os atributos do tipo dos objetos da VCN e suas Assertivas de Correspondência. Usamos $T_{[V]}$ para denotar o tipo dos objetos da VCN $\mathcal{V}$. Os atributos de $T_{[V]}$ são definidos de acordo com a estrutura dos elementos da VCN, conforme especificado no UID correspondente. De acordo com a estrutura dos elementos de $\mathcal{V}_{1,3}$ do UID $\mathcal{U}_1$ mostrado na Figura 4.1, $T_{[V1,3]}$ possui cinco atributos: título, data_publicação, conteudo, autor e mat_rel. No caso do atributo autor, cujo valor é um objeto de tipo estruturado denotado por $T_{[autor]}$, e do atributo mat_rel, cujo valor é um conjunto de objetos de tipo estruturado denotado por $T_{[mat\_rel]}$, deve-se especificar os atributos de $T_{[autor]}$ e $T_{[mat\_rel]}$. O símbolo ↘ ao lado dos atributos autor e mat_rel especifica que os mesmos possuem elos relacionados a eles (inf_autor e inf_mat_rel, respectivamente). Se necessário, deve ser incluído um atributo no tipo da VCN ou em um tipo estruturado definido na VCN, para identificar unicamente cada objeto do tipo. Dessa forma, é possível a passagem de parâmetros utilizando o objeto selecionado. As AC's dos atributos especificam como o valor de um atributo é derivado a partir do valor do atributo do objeto do tipo base. Por exemplo, suponha m_{V3} uma instância de $T_{[V3]}$, a qual corresponde ao objeto base m, instância do tipo $T_{materia}$. Da AC $\psi_2^{[V1,3]}$, temos que m_{V3}.titulo = m.titulo. Da AC $\psi_5^{[V3]}$, temos que m_{V3}.autor = m.autor.nome. Das AC's $\psi_6^{[V1,3]}$, $\psi_7^{[V1,3]}$ e $\psi_8^{[V1,3]}$, temos que m_{V3}.mat_rel = {struct {r.titulo, r.data_pub } | r ∈ m.mat_rel}.

- *Elos:* especifica os relacionamentos navegacionais da VCN. Um elo é uma conexão orientada entre duas VCN's, relacionando objetos de uma VCN com objetos de outra VCN.

Os parâmetros de um elo especificam a informação transportada para a VCN destino do elo. Por exemplo, o elo inf_autor permite navegar de uma matéria para o contexto da VCN $\mathcal{V}_4$ que contém as informações do autor da matéria. Nesse caso, o nome do autor da matéria é passado como parâmetro.

- *Operações:* relaciona as operações da VCN. Essas operações são um subconjunto das operações do tipo base da VCN.

- *Ordenação:* indica o conjunto de atributos usados para ordenar os objetos da VCN e o critério de ordenação (ascendente ou descendente) a ser utilizado. O critério *default* é o ascendente.

As especificações das VCN's podem ser realizadas de forma gráfica, semelhante à apresentada na seção 4.2.2, utilizando a interface da Ferramenta para Geração da Especificação de VCN. Suponha que um usuário deseja especificar a VCN $\mathcal{V}_{1,3}$. Primeiro, o usuário define, através de uma interface gráfica da ferramenta, o nome da VCN, seleciona a sua visão de objeto base, a partir de uma lista de visões de objeto criadas no ambiente, e, se necessário, define a condição de seleção. Essa ação gera a ACE da VCN, que especifica como os objetos da VCN são sintetizados a partir dos objetos da visão base. Para a VCN $\mathcal{V}_{1,3}$, a visão $T_{materias}$ é selecionada e a ACE $\psi_1^{[V1,3]}$:[$V_{1,3}$] ≡ [Materias_v[cod_mat=p]] é gerada, onde p é o parâmetro utilizado na condição de seleção.

Em seguida, o tipo da VCN deve ser definido e as ACC's desse tipo com o tipo da visão de objeto base devem ser geradas. Para gerar o tipo da VCN, a ferramenta apresenta uma tela contendo os atributos da visão de objeto base e o usuário seleciona, dentre esses atributos, os que pertencem à VCN. As ACC's do tipo gerado com o tipo da visão de objeto base são geradas pela ferramenta, automaticamente. Por exemplo, para definir o tipo da VCN $\mathcal{V}_{1,3}$, o usuário seleciona os atributos titulo, data_pub, conteudo, autor e mat_rel do tipo de objeto da visão de objeto base, ou seja, $T_{materia}$. No caso dos atributos estruturados autor e mat_rel, o usuário seleciona também os atributos do tipo estruturado que pertencem ao tipo da VCN. O tipo gerado é mostrado em formato de árvore de diretório e o usuário pode então selecionar os atributos do tipo que devem ser definidos como do tipo âncora. Nesse caso, aparece uma tela de criação de elo, onde é definido a VCN destino e os parâmetros

utilizados. Se a VCN ainda não existe, deverá ser criada. Para a VCN $\mathcal{V}_{1,3}$, o usuário define os atributos nome (de autor) e titulo (de mat_rel) como do tipo âncora. No caso do atributo autor, por exemplo, o elo inf_autor deve ser definido, onde a VCN destino é $\mathcal{V}_{1,4}$ e o parâmetro é self.autor.cod_aut. Note que o número de parâmetros e o tipo de cada um deles no elo da VCN de origem são iguais aos dos parâmetros utilizados pela VCN destino. O cartão de especificação de $\mathcal{V}_{1,3}$ gerado pela ferramenta é o apresentado na Figura 4.23.

Capítulo 5
Implementação e Manutenção das Visões de Contexto de Navegação

Neste Capítulo, discutimos as atividades Implementação e Manutenção das VCN's. Na atividade Implementação, as VCN's são implementadas a partir de suas especificações conceituais. É importante notar que a especificação conceitual das VCN's é totalmente independente do banco de dados e da plataforma de implementação. Neste trabalho, consideramos duas abordagens para implementação das VCN's: como visões de objeto e como visões XML. No primeiro caso, as VCN's são definidas através de consultas SQL sobre a sua visão de objetos base ou sobre o banco de dados. No último caso, os dados das VCN's extraídos do banco de dados são representados no formato XML. Como mostraremos neste Capítulo, o processo de implementação e manutenção de uma VCN pode ser realizado de forma automática, a partir de sua especificação e da especificação da sua visão de objeto base.

Este Capítulo está organizado da seguinte forma: na Seção 5.1, descrevemos o processo de implementação das VCN's utilizando visões de objetos; na Seção 5.2, descrevemos o processo de implementação das VCN's utilizando visões XML; na Seção 5.3, discutimos como é feita a manutenção das VCN's com relação a modificações no tipo de uma VCN e modificações no esquema do banco de dados.

5.1 Implementação como Visões de Objetos

Nesta abordagem, a definição da VCN consiste em uma consulta SQL que sintetiza os objetos da VCN a partir dos dados no banco de dados. A implementação da VCN pode ser feita em 3 ou 2 camadas, como mostrado nas Figuras 5.1 e 5.2, respectivamente. Na implementação em 3 camadas, são criadas as visões de objeto, e cada VCN é definida sobre a visão de objeto base. As visões de objetos são definidas por meio de consultas SQL sobre o esquema do banco de dados, ficando a cargo dessas visões resolver o problema de mapeamento de tuplas de tabelas (relacionais ou objeto-relacionais) em objetos de tipo complexo, cuja estrutura é compatível com a estrutura da VCN. Na implementação em 2 camadas, a consulta SQL que define a VCN é definida diretamente sobre o esquema do banco de dados.

A implementação em 3 camadas tem a vantagem de maior independência das VCN's em relação a modificações no esquema do banco de dados, uma vez que alterações no esquema do banco de dados afetam apenas a definição das visões de objetos. Entretanto, como nem todos os SGBD's disponíveis no mercado implementam o mecanismo de visões de objeto, consideramos, neste trabalho, as duas formas de implementação.

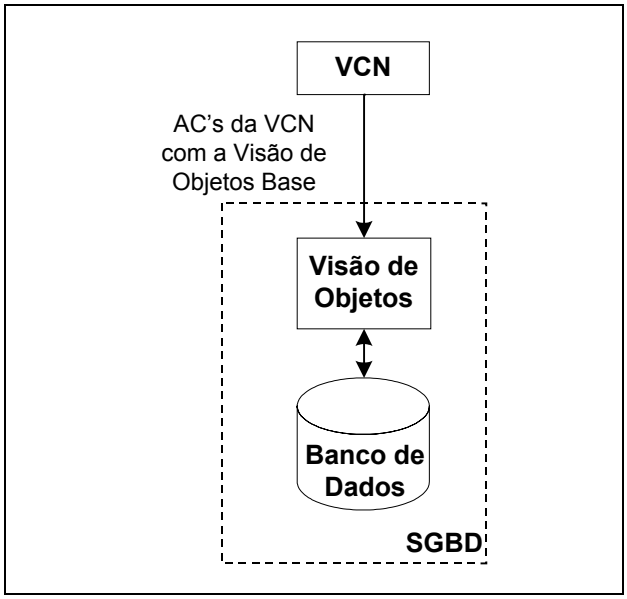


Figura 5.1: Implementação em 3 camadas

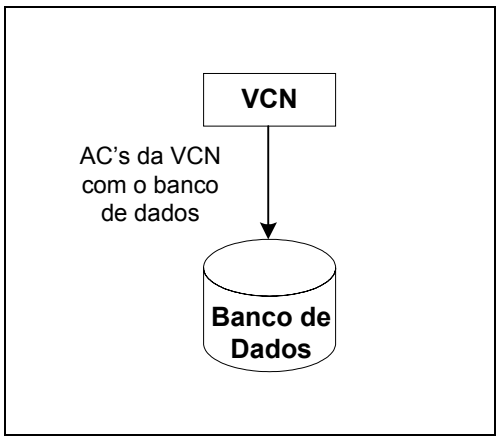


Figura 5.2: Implementação em 2 camadas.

5.1.1 Implementação em 3 camadas

Neste caso, as visões de objeto devem ser criadas usando o mecanismo discutido na seção 4.2.2. O mecanismo de visões de objeto permite definir visões lógicas, ricamente estruturadas, sobre um banco de dados já existente. No *Oracle* 9i, uma visão de objeto é definida através de uma consulta SQL:1999, que especifica como os objetos da visão são sintetizados a partir dos dados do banco de dados. No caso em que atualizações são permitidas, devem ser definidos tradutores (*instead of triggers*), os quais especificam como atualizações definidas sobre a visão de objeto são traduzidas em atualizações especificadas sobre o banco de dados. Nos exemplos deste Capítulo, considere o banco de dados relacional Publicações_rel da aplicação, mostrado na Figura 4.19, e o esquema das visões de objetos Publicações representado na 4.18.

As VCN's são definidas sobre o esquema da sua visão de objeto base através de uma consulta SQL, a qual mapeia os objetos da visão de objeto base em objetos da VCN. Essa consulta é simples, uma vez que a estrutura da visão de objeto base é compatível com a estrutura da VCN. O processo de implementação nesse enfoque consiste em dois passos, discutidos a seguir:

Passo 1: Criar as Visões de Objetos

Neste passo, para cada visão de objeto é gerada a consulta SQL que define a visão. A ferramenta VBA proposta em [42] permite gerar automaticamente, a partir do conjunto de assertivas de correspondência da visão, a consulta SQL que realiza o mapeamento definido pelas assertivas. No restante desta seção, considere as visões de objeto Materias_v e Autores_v do esquema Publicações, cujos cartões de especificação são mostrados na Figura 5.3 e na Figura 5.4, respectivamente. Cada cartão especifica a visão sobre o esquema do banco de dados Publicações_rel. Por exemplo, a consulta SQL que define a visão de objeto Materias_v é mostrada na Figura 5.5. Nessa consulta, a cláusula *from* é definida a partir da ACE ψ_1 e o identificador da visão é definido a partir da ACO ψ_2. Na cláusula *select*, o código SQL que define o valor de cada atributo dos objetos da visão é gerado a partir das ACC's da visão, como indicado na Figura 5.5.

Visão de Objeto: Materias_v	
Tabela Base: Materias_rel	(**ACE**: ψ_1: [Materias_v] ≡ [Materias_rel]) (**ACO:** ψ_2: [$T_{materia}$, {cod_mat}] ≡ [$T_{materias\_rel}$, {cod_mat}])
Atributos:	**ACC's:**
cod_mat: integer	ψ_3: [$T_{materia}$•cod_mat] ≡ [$T_{materias\_rel}$•cod_mat]
data_pub: string	ψ_4: [$T_{materia}$•data_pub] ≡ [$T_{materias\_rel}$•data_pub]
titulo: string	ψ_5: [$T_{materia}$•titulo] ≡ [$T_{materias\_rel}$•titulo]
resumo: string	ψ_6: [$T_{materia}$•resumo] ≡ [$T_{materias\_rel}$•resumo]
conteudo: string	ψ_7 : [$T_{materia}$•conteudo] ≡ [$T_{materias\_rel}$•conteudo]
autor: REF <<T_{autor}>>	ψ_8 : [$T_{materia}$•autor] ≡ [$T_{materias\_rel}$•fk_1]
mat_rel: set of REF <<$T_{materia}$>>	ψ_9 : [$T_{materia}$•mat_rel] ≡ [$T_{materias\_rel}$•fk_2^{-1}•fk_3]

Figura 5.3: Cartão de especificação da visão de objeto Materias_v

Visão de Objeto: Autores_v	
Tabela Base: Autores_rel	(**ACE**: ψ_{10}: [Autores_v] ≡ [Autores_rel]) (**ACO:** ψ_{11}: [T_{autor}, {cod_aut}] ≡ [T_{autor_rel}, {cod_aut}])
Atributos:	**ACC's:**
cod_aut: integer	ψ_{12}: [T_{autor}•cod_aut] ≡ [$T_{autores\_rel}$•cod_aut]
nome: string	ψ_{13}: [T_{autor}•nome] ≡ [$T_{autores\_rel}$•nome]
foto: string	ψ_{14}: [T_{autor}•foto] ≡ [$T_{autores\_rel}$•foto]
biografia: string	ψ_{15}: [T_{autor}•biografia] ≡ [$T_{autores\_rel}$•biografia]

Figura 5.4: Cartão de especificação da visão de objeto Autores_v

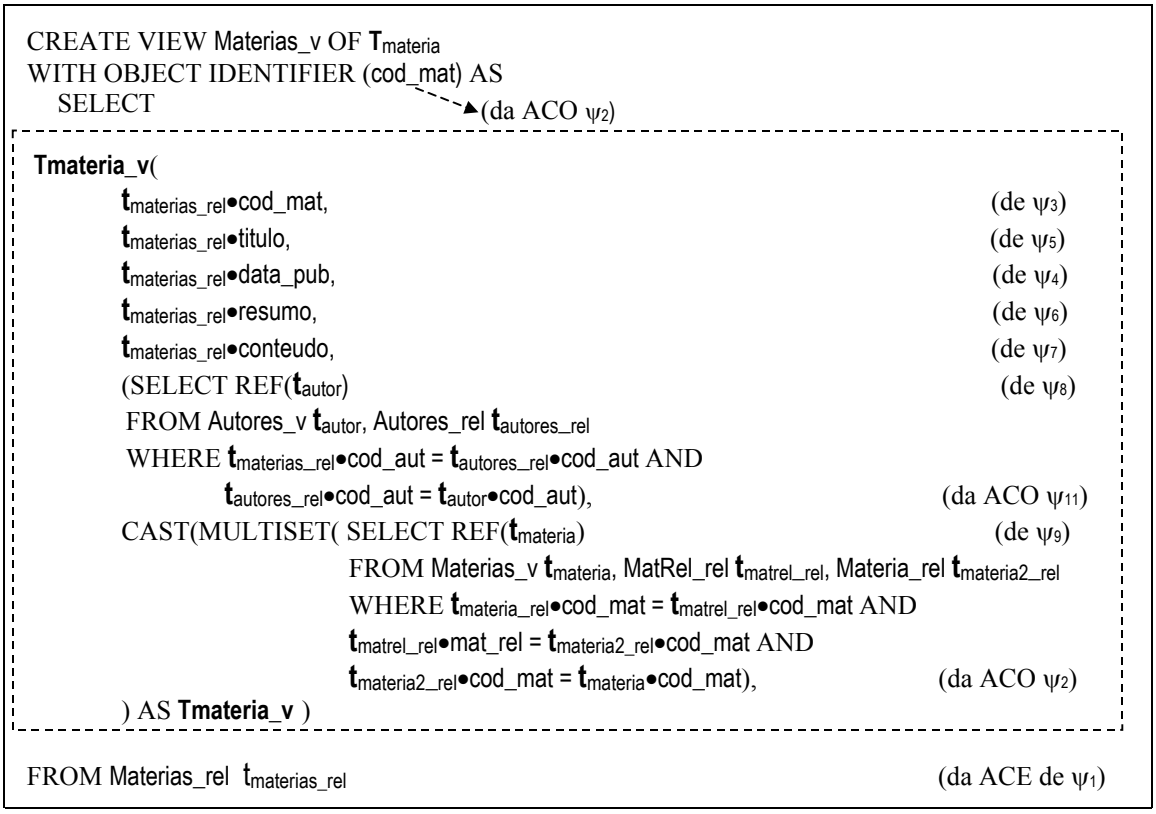


Figura 5.5: Definição da Visão de Objeto Materias_v

<u>Passo 2</u>: Gerar definição da VCN sobre esquema da visão de objeto base

Neste passo, para cada VCN é gerada a consulta SQL, a qual mapeia os objetos da visão de objeto base em objetos da VCN. No ambiente proposto, essa consulta pode ser gerada de forma automática com a Ferramenta de Geração de Consultas. No Capítulo 6, é discutido o algoritmo que gera a consulta SQL que define uma VCN, a partir das AC's da VCN com a visão de objeto base e os critérios de ordenação, conforme especificado no cartão de especificação da VCN. A existência de parâmetros faz gerar uma consulta parametrizada. A definição da VCN pode ser armazenada no banco de dados, na forma de procedimentos armazenados. A Figura 5.7 mostra a definição da VCN $\mathcal{V}_{1,3}$ do UID $\mathcal{U}_1$ (vide Figura 4.1), especificada na Figura 5.6, e as AC's da VCN, usadas na geração da consulta SQL.

VCN: $\mathcal{V}_{1,3}$	
Parâmetros: param : integer	
Visão de Objeto Base: Materias_v	**Condição de Seleção** (**ACE**: $\psi_1^{[V13]}$: $[V_{1,3}] \equiv$ [[Materias_v[cod_mat = param]]) cod_mat = param
Atributos:	**Assertivas de Correspondência**
titulo: string	$\psi_2^{[V1,3]}$: $[T_{[V1,3]} \bullet titulo] \equiv [Tmateria \bullet titulo]$
data_publicação: string	$\psi_3^{[V1,3]}$: $[T_{[V1,3]} \bullet data\_publicação] \equiv [Tmateria \bullet data\_pub]$
conteudo: string	$\psi_4^{[V1,3]}$: $[T_{[V1,3]} \bullet conteudo] \equiv [Tmateria \bullet conteudo]$
↘ autor: <$T_{[autor]}$>	$\psi_5^{[V1,3]}$: $[T_{[V1,3]} \bullet autor] \equiv [Tmateria \bullet autor]$
cod_aut: integer	$\psi_6^{[V1,3]}$: $[T_{[autor]} \bullet cod\_aut] \equiv [Tautor \bullet cod\_aut]$
nome: string	$\psi_7^{[V1,3]}$: $[T_{[autor]} \bullet nome] \equiv [Tautor \bullet nome]$
↘ mat_rel: set of <$T_{[mat\_rel]}$>	$\psi_8^{[V1,3]}$: $[T_{[V1,3]} \bullet mat\_rel] \equiv [Tmateria \bullet mat\_rel]$
cod_mat: integer	$\psi_9^{[V1,3]}$: $[T_{[mat\_rel]} \bullet cod\_mat] \equiv [Tmateria \bullet cod\_mat]$
titulo: string	$\psi_{10}^{[V1,3]}$: $[T_{[mat\_rel]} \bullet titulo] \equiv [Tmateria \bullet titulo]$
data_publicação: string	$\psi_{11}^{[V1,3]}$: $[T_{[mat\_rel]} \bullet data\_publicação] \equiv [Tmateria \bullet data\_pub]$
Elos: inf_autor: {**Destino**: $V_{1,4}$; **Parâmetros**:{self.autor.cod_aut}} inf_mat_rel:{**Destino**:$V_{1,3}$; **Parâmetros**:{self.mat.cod_mat | mat ∈ mat_rel}}	
Operações: imprimirMateria() enviarMateria()	
Ordenação:	

Figura 5.6: Cartão de Especificação da VCN $\mathcal{V}_{1,3}$

```
CREATE OR REPLACE PROCEDURE V13
(param IN VARCHAR )  ····► parâmetros
AS
BEGIN
1.  SELECT  t_materia•titulo AS titulo,  ····► ψ2[V1,3]
2.          t_materia•data_pub AS data_publicação,  ····► ψ3[V1,3]
3.          t_materia•conteudo AS conteudo,  ····► ψ4[V1,3]
4.          CURSOR ( SELECT t_materia•autor•cod_aut AS cod_aut,
5.                       t_materia•autor•nome AS nome
6.                       FROM dual) AS autor  ····► ψ5[V1,3]
7.          CURSOR ( SELECT t_materia2•COLUMN_VALUE•cod_mat AS cod_mat,
8.                       t_materia2•COLUMN_VALUE•titulo AS titulo,
9.                       t_materia2•COLUMN_VALUE•data_pub AS data_publicação  ····► ψ8[V1,3]
10.                      FROM TABLE(t_materia•mat_rel) t_materia2 )
FROM Materias_v t_materia
WHERE t_materia•cod_mat = param  ····► ψ1[V1,3]
END;
```

Figura 5.7: Definição da VCN $\mathcal{V}_{1,3}$ sobre esquema da visão de objeto Materias_v

5.1.2 Implementação em 2 camadas

Na implementação em 2 camadas, as VCN's são definidas através de consultas SQL especificadas sobre o esquema do banco de dados, a qual mapeia os objetos do banco de dados em objetos da VCN. Para gerar essas consultas, as AC's da VCN com o esquema do banco de dados devem ser derivadas a partir das AC's da VCN com o esquema da visão de objeto base e das AC's dessa visão com o esquema do banco de dados. A desvantagem dessa implementação é que alterações no banco de dados podem requerer a reimplementação de várias VCN's. O processo de implementação das VCN's consiste em dois passos, discutidos a seguir:

<u>Passo 1</u>: Gerar Assertivas de Correspondência da VCN com o esquema do banco de dados

Neste passo, as AC's da VCN com o banco de dados são inferidas a partir da composição das AC's da VCN com a visão de objeto base e das AC's da visão de objeto base com o banco de dados. Por exemplo, das ACE's $\psi_1^{[V1,3]}$: **[V**$_{1,3}$**]** ≡ **[Materias_v[cod_mat = param]]** e ψ_1: **[Materias_v]** ≡ **[Materias_rel],** pode-se inferir a ACE $\delta_1^{[V1,3]}$: **[V**$_{1,3}$**]** ≡ **[Materias_rel[cod_mat = param]]**. Das ACC's $\psi_5^{[V1,3]}$ e ψ_8, obtemos a ACC $\delta_5^{[V1,3]}$: [$T_{[V1,3]}$.autor] ≡ [$T_{materias\_rel}$•fk_1]. O novo cartão de especificação de $\mathcal{V}_{1,3}$ é mostrado na Figura 5.8, que lista as ACC's de $\mathbf{T}_{[V1,3]}$&$\mathbf{T}_{materias\_rel}$ obtidas da composição das ACC's de $\mathbf{T}_{materia}$ & $\mathbf{T}_{materias\_rel}$ (Figura 5.3) e das AC's de $\mathbf{T}_{[V1,3]}$ & $\mathbf{T}_{materia}$ (Figura 5.6). No ambiente proposto, as AC's das VCN's com o banco de dados são inferidas de forma automática. O algoritmo de derivação de assertivas, denominado GeraAssertivas, é apresentado na seção 6.3.

<u>Passo 2</u>: Gerar definição da VCN sobre esquema do banco de dados

Neste passo, para cada VCN é gerada a consulta SQL que define a VCN, como descrito no passo 2 da seção 5.1.1. Geralmente, essa consulta é mais complexa do que a consulta gerada na implementação em 3 camadas, pois a estrutura dos objetos da VCN (atributos e relacionamentos) pode ser bem diferente da estrutura dos objetos do banco de dados. A Figura 5.9 mostra a definição da VCN $\mathcal{V}_{1,3}$ do UID $\mathcal{U}_1$ sobre o esquema do banco de dados e as AC's usadas na geração da consulta SQL.

VCN: $\mathcal{V}_{1,3}$		
Parâmetros: param : integer		
Tabela Base: Materias_rel	**Condição de Seleção** cod_mat = p	(**ACE**: $\delta_1^{[V1,3]}$: $[V_{1,3}] \equiv$ [Materias_rel[cod_mat = param]])
Atributos:	**Assertivas de Correspondência**	
titulo: string	$\delta_2^{[V1,3]}$: $[T_{[V1,3]} \bullet \text{titulo}] \equiv [T_{\text{materias\_rel}} \bullet \text{titulo}]$ (de $\psi_2^{[V1,3]}$ ψ_5)	
data_publicação: string	$\delta_3^{[V1,3]}$: $[T_{[V1,3]} \bullet \text{data\_publicação}] \equiv [T_{\text{materias\_rel}} \bullet \text{data\_pub}]$ (de $\psi_3^{[V1,3]}$ e ψ_4)	
conteudo: string	$\delta_4^{[V1,3]}$: $[T_{[V1,3]} \bullet \text{conteudo}] \equiv [T_{\text{materias\_rel}} \bullet \text{conteudo}]$ (de $\psi_4^{[V1,3]}$ e ψ_7)	
↘ autor: <$T_{[autor]}$>	$\delta_5^{[V1,3]}$: $[T_{[V1,3]} \bullet \text{autor}] \equiv [T_{\text{materias\_rel}} \bullet fk_1]$ (de $\psi_5^{[V1,3]}$ e ψ_8)	
cod_aut: integer	$\delta_6^{[V1,3]}$: $[T_{[autor]} \bullet \text{cod\_aut}] \equiv [T_{\text{autores\_rel}} \bullet \text{cod\_aut}]$ (de $\psi_6^{[V1,3]}$ e ψ_{12})	
nome: string	$\delta_7^{[V1,3]}$: $[T_{[autor]} \bullet \text{nome}] \equiv [T_{\text{autores\_rel}} \bullet \text{nome}]$ (de $\psi_7^{[V1,3]}$ e ψ_{13})	
↘ mat_rel: set of <$T_{[mat\_rel]}$>	$\delta_8^{[V1,3]}$: $[T_{[V1,3]} \bullet \text{mat\_rel}] \equiv [T_{\text{materias\_rel}} \bullet fk_2^{-1} \bullet fk_3]$ (de $\psi_8^{[V1,3]}$ e ψ_9)	
cod_mat: integer	$\delta_9^{[V1,3]}$: $[T_{[mat\_rel]} \bullet \text{cod\_mat}] \equiv [T_{\text{materias\_rel}} \bullet \text{cod\_mat}]$ (de $\psi_9^{[V1,3]}$ e ψ_3)	
titulo: string	$\delta_{10}^{[V1,3]}$: $[T_{[mat\_rel]} \bullet \text{titulo}] \equiv [T_{\text{materias\_rel}} \bullet \text{titulo}]$ (de $\psi_{10}^{[V1,3]}$ e ψ_5)	
data_publicação: string	$\delta_{11}^{[V1,3]}$: $[T_{[mat\_rel]} \bullet \text{data\_publicação}] \equiv [T_{\text{materias\_rel}} \bullet \text{data\_pub}]$ (de $\psi_{11}^{[V1,3]}$ e ψ_4)	
Elos:		
inf_autor: {**Destino**: $V_{1,4}$; **Parâmetros**:{self.autor.cod_aut}}		
inf_mat_rel:{**Destino**:$V_{1,3}$; **Parâmetros**:{self.mat.cod_mat | mat ∈ mat_rel}}		
Operações:		
imprimirMateria()		
enviarMateria()		
Ordenação:		

Figura 5.8: Cartão de Especificação da VCN $\mathcal{V}_{1,3}$, especificada sobre o banco de dados

```
CREATE OR REPLACE PROCEDURE V13
(param IN VARCHAR ) AS
BEGIN
1.  SELECT  t_materias_rel•titulo AS titulo,                          → δ2[V1,3]
2.          t_materias_rel•data_pub AS data_publicacao,               → δ3[V1,3]
3.          t_materias_rel•conteudo AS conteudo,                      → δ4[V1,3]
4.          CURSOR ( SELECT  t_autores_rel•cod_aut AS cod_aut,
5.                           t_autores_rel•nome AS nome               → δ5[V1,3]
6.                   FROM Autores_rel t_autores_rel
7.                   WHERE t_materias_rel•cod_aut= t_autores_rel•cod_aut) as autor,
8.          CURSOR ( SELECT  t_materias_rel2•cod_mat AS cod_mat,
9.                           t_materias_rel2•titulo AS titulo,
10.                          t_materias_rel2• data_pub AS data_publicacao  → δ8[V1,3]
11.                  FROM Materias_rel t_materias_rel2, MatRel_rel t_matRel_rel)
12.                  WHERE t_matRel_rel•mat_rel = t_materias_rel2•cod_mat
13.                  AND t_matRel_rel•cod_mat = t_materias_rel•cod_mat ) as mat_rel
FROM Materias_rel t_materias_rel
WHERE t_materias_rel•cod_mat = param                                  → δ1[V1,3]
END;
```

Figura 5.9: Definição da VCN $\mathcal{V}_{1,3}$ sobre esquema do banco de dados

5.2 Implementação como Visões XML

Diversas aplicações existentes utilizam a *web* como canal de comunicação entre as empresas e os clientes [7]. O crescimento de transações *business-to-business* (B2B) via *Internet* fez surgir a necessidade das empresas por aplicações que possibilitem a troca em tempo real de informações que conduzem os negócios da empresa. Isso significa, por exemplo, que catálogos de venda *on-line* devem consultar um banco de dados para listar os produtos disponíveis e, além disso, devem verificar autorização de crédito e processamento do pedido. Para satisfazer essa nova demanda, é necessária uma infraestrutura que apóie a parceria de empresas para viabilizar a troca de informações. A linguagem XML tem se tornado uma solução para essas necessidades, pois possibilita a definição de um esquema único para troca de informações. O uso de visões XML possibilita a publicação de dados armazenados em bases de dados convencionais no formato XML.

Na implementação como visões XML, os dados das VCN's extraídos do banco de dados são representados no formato XML. Neste trabalho, investigamos três tipos de *framework* para publicação dos dados das VCN's no formato XML, onde os dados estão armazenados em bases de dados relacionais ou objeto-relacionais. Conforme discutido na seção 3.5, dois desses enfoques são baseados na utilização de *middlewares*, que foram desenvolvidos para permitir a publicação de dados armazenados em bases convencionais no formato XML [6][46][47][19][20][42]. O terceiro enfoque é baseado na geração de páginas que extraem os dados do banco de dados e transformam o resultado no formato XML.

5.2.1 Publicação das VCN's no *XPeranto*

Em sistemas como o *Silkroute* [19][20] e o *XPeranto* [6][46][47], conforme foi descrito nas seções 4.5.1 e 4.5.2, as tabelas do banco de dados são apresentadas para o projetista como uma visão XML *Default*. A Figura 5.10 mostra a visão XML *Default* do esquema do banco de dados Publicações_rel. Usuários podem então definir suas próprias visões XML no topo dessa visão *default*, ou simplesmente submeter consultas XQuery ao sistema. Uma visão XML é definida através de uma consulta XQuery que mapeia a visão

XML *Default* na visão XML desejada. Consultas XQuery, quando submetidas ao *XPeranto* ou ao *Silkroute*, são traduzidas em consultas SQL sobre o banco de dados.

O processo de implementação da VCN, segundo este enfoque, consiste em quatro passos, discutidos a seguir.

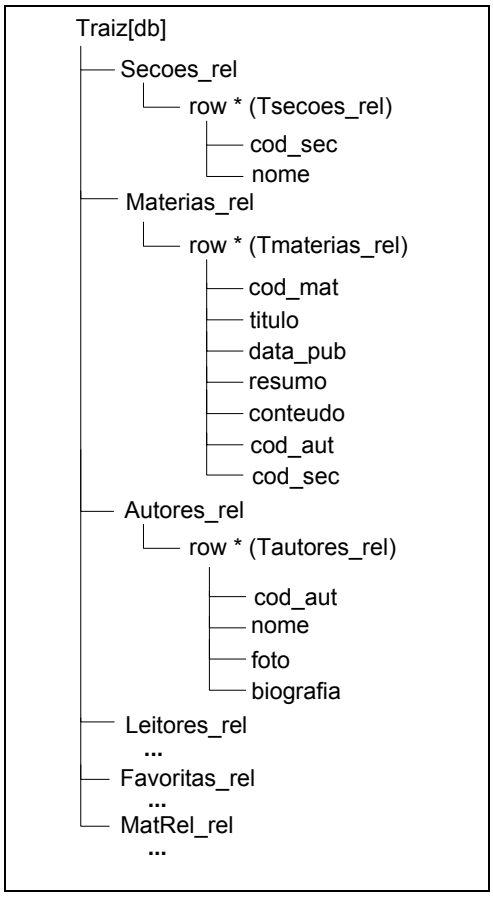


Figura 5.10: Visão XML *Default* do banco de dados Publicações_rel

<u>Passo 1</u>: Gerar esquema da visão XML da aplicação e suas Assertivas de Correspondência

Neste passo, é definido o esquema da visão XML da aplicação. Para isso, é feito o mapeamento das visões de objetos no esquema XML da visão XML da aplicação. Para representar a visão XML da aplicação, utilizamos a notação gráfica descrita na seção 3.2.1.4. Em [7] são apresentadas regras para mapear um esquema orientado a objetos em um esquema XML. Existem também algumas ferramentas [17][39] que automatizam esse processo de mapeamento.

A Figura 5.11 mostra o esquema XML que representa as visões de objeto da aplicação Publicações. O tipo Traiz[Publicações] contém um elemento primário para cada visão de objeto em Publicações. Observe que os atributos monovalorados e multivalorados dos tipos das visões de objeto são diretamente mapeados em elementos monocorrência e multiocorrência da visão XML. Os atributos de referência são mapeados em elementos de referência no esquema XML. Por exemplo, o atributo multivalorado de referência mat_rel de $T_{materia}$ é mapeado no elemento multiocorrência mat_rel, que é uma referência para $T_{materia}$.

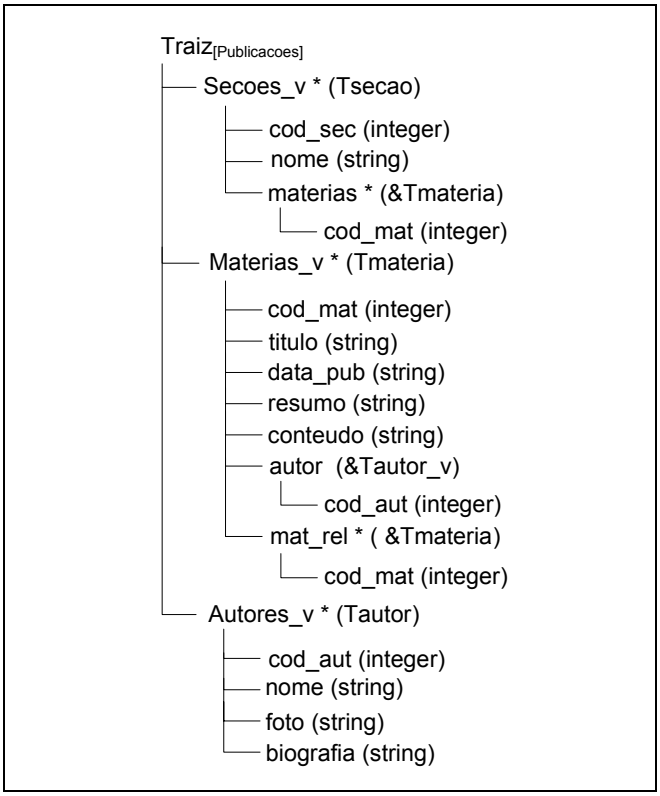


Figura 5.11: Esquema da visão XML da aplicação Publicações

As AC's entre as visões de objeto e o banco de dados são convertidas diretamente em AC's da visão XML com a visão XML *Default*. Usamos o conjunto de assertivas de correspondência proposto em [5] para especificar as correspondências entre esquemas XML. As Assertivas de Correspondência de Coleções Globais (ACCG) especificam as correspondencias entre o esquema da visão XML da aplicação e esquema da visão XML *Default*. As Assertivas de Correspondência de Caminho (ACC) especificam como sintetizar os elementos da visão a partir das tuplas do banco de dados. As Assertivas de

Correspondência de Elementos (ACE) especificam sob que condições dois elementos representam o mesmo objeto do mundo real, ou seja, são semanticamente equivalentes.

Considere, por exemplo, o esquema do banco de dados Publicações_rel. O esquema da visãoXML *Default* de Publicações_rel gerada pelo *XPeranto* é mostrada na Figura 5.10. Na Figura 5.12, mostramos as AC's da visão XML da aplicação com a visão XML *Default*.

ACCG de Secoes_v & Secoes_rel
θ_1: [Traiz$_{[publicacoes]}$/Secoes_v] ≡ [Traiz$_{[db]}$/Secoes_rel]

ACC's de $\mathbf{T}_{secao}$ & $\mathbf{T}_{secoes\_rel}$
θ_4: [$\mathbf{T}_{secao}$/cod_sec] ≡ [$\mathbf{T}_{secoes\_rel}$/cod_sec]
θ_5: [$\mathbf{T}_{secao}$/nome] ≡ [$\mathbf{T}_{secoes\_rel}$/nome]
θ_6: [$\mathbf{T}_{secao}$/materias] ≡ [$\mathbf{T}_{secoes\_rel}$/fk_4^{-1}]

ACCG de Materias_v & Materias_rel
θ_2: [Traiz$_{[publicacoes]}$/Materias_v] ≡ [Traiz$_{[db]}$/Materias_rel]

ACC's de $\mathbf{T}_{materia}$ & $\mathbf{T}_{materias\_rel}$
θ_7: [$\mathbf{T}_{materia}$ /cod_mat}] ≡ [$\mathbf{T}_{materias\_rel}$/cod_mat]
θ_7: [$\mathbf{T}_{materia}$ /titulo}] ≡ [$\mathbf{T}_{materias\_rel}$/titulo]
θ_8: [$\mathbf{T}_{materia}$ /data_pub] ≡ [$\mathbf{T}_{materias\_rel}$/data_pub}]
θ_{10}: [$\mathbf{T}_{materia}$ /resumo] ≡ [$\mathbf{T}_{materias\_rel}$/resumo}]
θ_{11}: [$\mathbf{T}_{materia}$ /conteudo] ≡ [$\mathbf{T}_{materias\_rel}$/conteudo}]
θ_{12}: [$\mathbf{T}_{materia}$ /autor] ≡ [$\mathbf{T}_{materias\_rel}$/$fk_1$]
θ_{13}: [$\mathbf{T}_{materia}$/mat_rel] ≡ [$\mathbf{T}_{materias\_rel}$/$fk_2^{-1}$/ fk_3]

ACCG de Autores_v & Autores_rel
θ_3: [Traiz$_{[publicacoes]}$/Autores_v] ≡ [Traiz$_{[db]}$/Autores_rel]

ACC's de $\mathbf{T}_{autor}$ & $\mathbf{T}_{autores\_rel}$
θ_{14}: [$\mathbf{T}_{autor}$/cod_aut] ≡ [$\mathbf{T}_{autores\_rel}$/cod_aut]
θ_{15}: [$\mathbf{T}_{autor}$/nome] ≡ [$\mathbf{T}_{autores\_rel}$/nome]
θ_{16}: [$\mathbf{T}_{autor}$/foto] ≡ [$\mathbf{T}_{autores\_rel}$/foto]
θ_{17}: [$\mathbf{T}_{autor}$ /biografia] ≡ [$\mathbf{T}_{autores\_rel}$/biografia]

Figura 5.12: Assertivas de Correspondência - Visão XML da aplicação & Visão XML *Default*

<u>Passo 2</u>: Gerar definição da Visão XML da aplicação

Neste passo, a visão XML da aplicação é definida através de uma consulta XQuery sobre a Visão XML *Default* do banco de dados. Esta consulta pode ser gerada de forma automática a partir das assertivas de correspondência entre o esquema da visão XML da aplicação com o esquema da visão XML *Default*, obtidas no passo 1. Na Figura 5.13,

mostramos a definição da Visão XML da Figura 5.11 sobre a visão XML *Default* do banco de dados.

Passo 3: Gerar definição da VCN sobre visão XML da aplicação

Neste passo, as VCN's são definidas através de uma consulta XQuery sobre o esquema da visão XML da aplicação. A consulta XQuery pode ser gerada de forma automática a partir da especificação conceitual da VCN. Para isso, o algoritmo para geração da consulta deve usar a condição de seleção, as assertivas de correspondências da VCN com a visão de objeto base e os critérios de ordenação conforme especificado no cartão de especificação da VCN. A existência de parâmetros faz gerar uma consulta parametrizada. As consultas XQuery que definem as VCN's $\mathcal{V}_{1,2}$ e $\mathcal{V}_{1,3}$ do UID $\mathcal{U}_1$ (vide Figura 4.1) são mostradas na Figura 5.14 e Figura 5.15, respectivamente. A consulta XQuery que define $\mathcal{V}_{1,2}$ obtém todas as matérias que pertencem à seção selecionada na interação de $\mathcal{V}_{1,1}$. A consulta XQuery que define $\mathcal{V}_{1,3}$ obtém os detalhes da matéria selecionada na interação de $\mathcal{V}_{1,2}$. Nesse caso, titulo é passado como parâmetro para a consulta de $\mathcal{V}_{1,3}$.

A implementação das VCN's utilizando esse tipo de *middleware* apresenta algumas limitações: (i) aplica-se somente a bancos de dados relacionais; (ii) existe uma forte dependência entre a Visão XML *Default* e o banco de dados, de modo que uma mudança no banco de dados requer a redefinição da Visão XML da aplicação; (iii) a consulta XQuery que mapeia os elementos da Visão XML *Default* em elementos da Visão XML da aplicação geralmente é complexa, o que requer conhecimentos avançados de XQuery do projetista; e (iv) não há garantia de processamento eficiente das consultas pelos *middlewares* [26]. No entanto, o processo de geração das consultas XQuery que definem as visões XML e as VCN's pode ser automatizado, como discutido nos passos 2 e 3, a partir das especificações da VCN e da sua visão de objeto base.

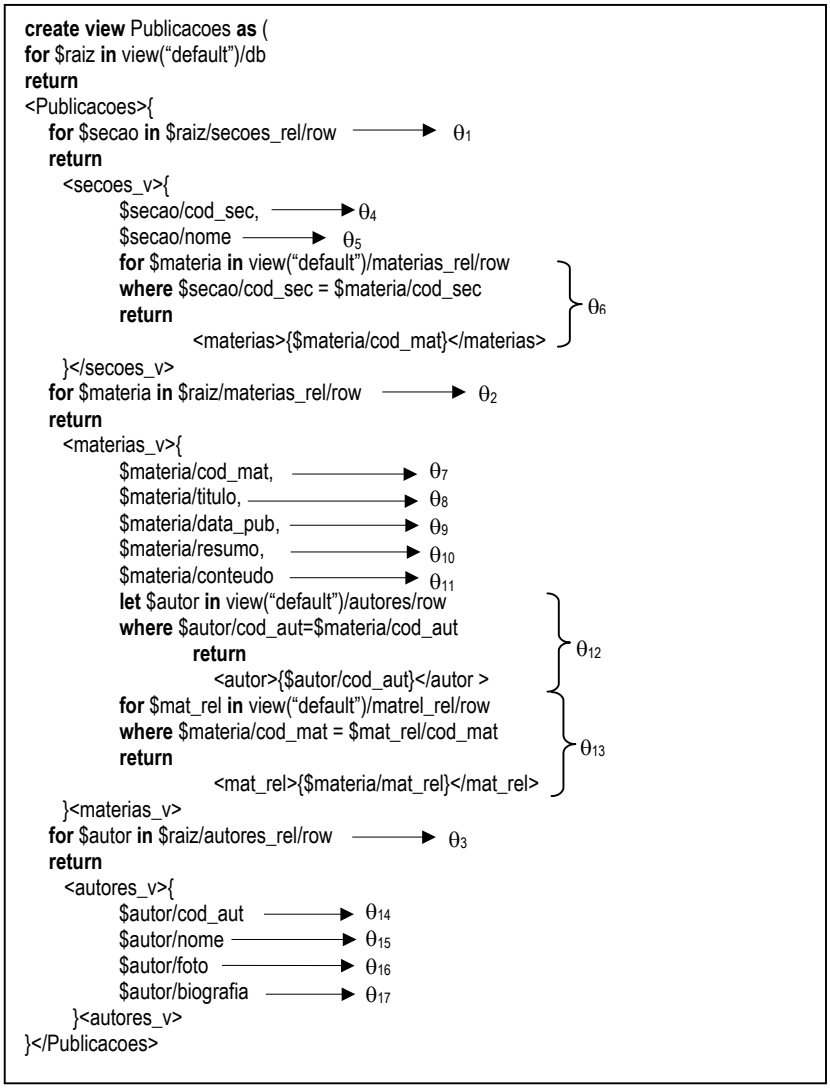


Figura 5.13: Definição da visão XML da aplicação Publicações

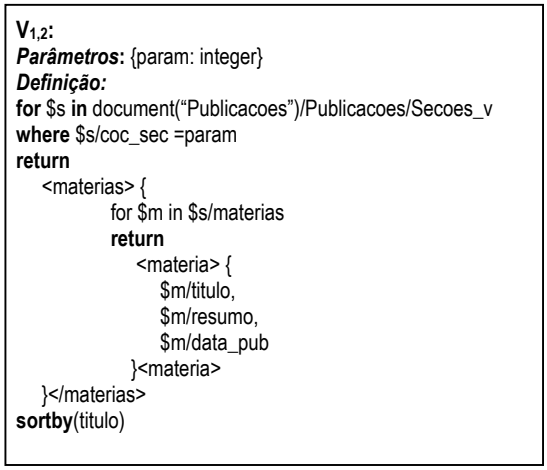


Figura 5.14: Definição da VCN $\mathcal{V}_{1,2}$

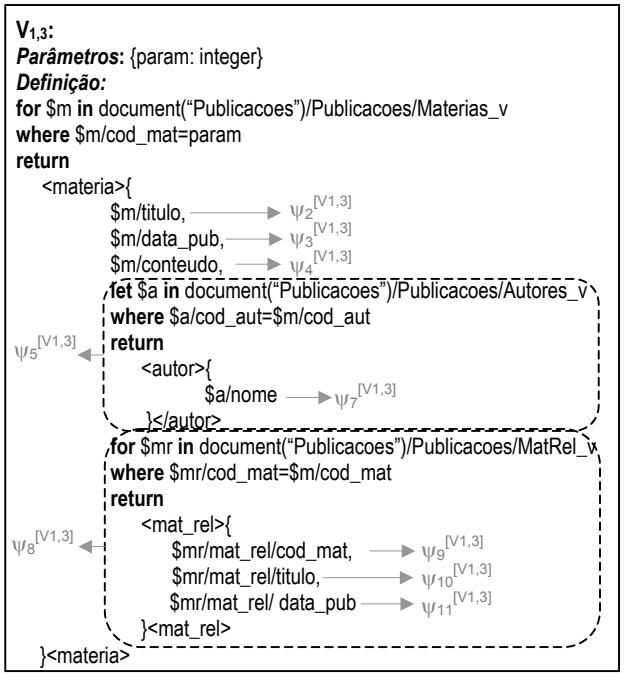


Figura 5.15: Definição da VCN $\mathcal{V}_{1,3}$

5.2.2 Publicação das VCN's no XML *Publisher*

No XML *Publisher* [42], conforme foi descrito na seção 3.5.3, para cada visão XML publicada no XML *Publisher* existe uma visão de objetos associada, chamada de visão de objetos *default* (VOD). Uma consulta XQuery, quando submetida ao sistema, é traduzida em uma consulta SQL:1999 aplicada à respectiva VOD. Em seguida, a consulta SQL gerada é processada pelo SGBD, com base na definição da VOD. Finalmente, o resultado da consulta SQL, que é uma coleção de objetos, deverá ser manipulado no XML *Publisher* para gerar a resposta XML especificada na consulta XQuery de entrada.

O processo de implementação das VCN's, segundo este enfoque, consiste em três passos, discutidos a seguir:

Passo 1: Gerar esquema XML da aplicação

A geração do esquema XML da Visão do UID é feita da mesma forma que foi apresentado no passo 1 da seção 5.2.1.

Passo 2: Criar Visão de Objetos

Deve-se gerar as visões de objetos da visão XML. O processo de geração das visões de objetos é feito da mesma forma que foi apresentado no passo 1 da seção 5.1.1.

Passo 3: Gerar Definição da VCN sobre visão XML da aplicação

A definição de cada VCN através de uma consulta XQuery é gerada como foi descrito no passo 3 da seção 5.2.1. As consultas XQuery que definem as VCN's são processadas pelo *XQuery Translator*, onde são reformuladas em consultas SQL sobre a visão de objetos. Da mesma forma que as consultas formuladas no enfoque apresentado na seção 5.1.1, as consultas traduzidas pelo *XQuery Translator* são consultas simples, pois são especificadas sobre o esquema XML da aplicação, o qual possui estrutura compatível com as visões de objetos.

Além das vantagens da utilização de visões de objetos, já discutidas na seção 5.1.1, esse enfoque se aplica também a publicação de dados armazenados em banco de dados objeto-relacionais, enquanto que o enfoque anterior se limita à publicação de banco de dados relacionais. No entanto, esse enfoque só pode ser utilizado quando o SGBD em uso suporta visões de objetos, como por exemplo, o *Oracle* 9i.

5.2.3 Publicação das VCN's através de páginas dinâmicas

A implementação das VCN's nesse enfoque é feita através da construção de páginas dinâmicas, uma para cada VCN. Essas páginas são armazenadas em módulos tradutores que residem na camada mediadora do servidor *Web* e contêm as consultas SQL que extraem do banco de dados os dados requeridos pelas VCN's. As páginas podem ser implementadas utilizando *scripts* ou módulos compilados. Módulos compilados são códigos binários executados no servidor, os quais produzem páginas HTML que são enviadas ao cliente requisitante. Algumas implementações mais populares dessa categoria são *Internet Server* API (ISAPI) [31] e *Java Servlets* [48]. Páginas que usam *server-side scripts* são armazenadas no servidor *web* e contêm *scripts* que são interpretados por um processador apropriado no servidor de aplicações. O resultado é uma página HTML formatada que é

enviada ao cliente requisitante. Algumas implementações mais populares oferecidas nessa categoria são *Java Server Pages* (JSP) [49], *Active Server Pages* (ASP) [31] e PHP [37].

Neste trabalho, a implementação das VCN's é exemplificada através do uso de XSQL *Pages*. Como já foi apresentado na seção 3.5.4, XSQL *pages* permite publicar conteúdo dinâmico na *web* no formato XML a partir de dados extraídos de bases relacionais ou objeto-relacionais. A Figura 5.16 mostra a XSQL *Pages* criada para a VCN $\mathcal{V}_{1,3}$. As páginas XSQL possuem associadas folhas de estilo XSLT que permitem transformar o documento XML canônico, resultado do processamento da página, no formato do esquema XML da VCN correspondente.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="V13.xsl"?>
<xsql:query connection="xml" xmlns:xsql="urn:oracle-xsql">
      SELECT  t_materia_rel•titulo AS titulo,
              t_materia_rel•data_pub AS data_publicacao,
              t_materia_rel•conteudo AS conteudo,
              CURSOR ( SELECT  t_autor_rel•cod_aut AS cod_aut,
                               t_autor_rel•cod_nome AS nome
                       FROM Autores_rel t_autor_rel
                       WHERE t_materia_rel•cod_aut= t_autor_rel•cod_aut) as autor
              CURSOR ( SELECT  t_materia_rel2•cod_mat AS cod_mat,
                               t_materia_rel2•titulo AS titulo,
                               t_materia_rel2• data_pub AS data_publicacao
                       FROM Materias_rel t_materia_rel2, MatRel_rel t_matRel_rel)
                       WHERE t_matRel_rel•mat_rel = t_materia_rel2•cod_mat
                       AND t_matRel_rel•cod_mat = t_materia_rel•cod_mat ) as mat_rel
FROM Materias_rel t_materia_rel
WHERE t_materia_rel•cod_mat = {@param}
</xsql:query>
```

Figura 5.16: Página XSQL gerada para $\mathcal{V}_{1,3}$

Essa solução é bastante eficiente, mas se torna complexa para o caso de aplicações DIWA, que geralmente envolve um grande número de VCN's e, assim, uma grande quantidade de páginas necessita ser gerada. Além disso, alterações no esquema do banco de dados podem requerer a re-implementação de várias VCN's. Uma das contribuições deste

trabalho é automatizar o processo de geração e manutenção das páginas XSQL que implementam as VCN's. Esse processo consiste em três passos, discutidos a seguir:

Passo 1: Gerar consulta SQL que define a VCN

Neste passo, é gerada a consulta SQL que define a VCN. Como discutido na seção 5.2, a consulta SQL pode ser definida de duas formas: (i) sobre o esquema da visão de objetos do UID, ou (ii) sobre o esquema do banco de dados. Em ambos os casos, a consulta SQL é gerada de modo automático, como discutido nas seções 5.1.1 e 5.1.2, respectivamente. Note que a consulta contida na tag <xsql:query> da Figura 5.16, que corresponde à definição da VCN $\mathcal{V}_{1,3}$ sobre o banco de dados Publicações_rel, é a mesma definida no procedimento armazenado da Figura 5.9.

Passo 2: Gerar Esquema XML da VCN

Neste passo, o esquema XML da VCN é gerado de acordo a estrutura da VCN. A partir da estrutura Matéria(titulo, data de publicação, Autor(nome), Materias Relacionadas (titulo, data de publicação) da VCN $\mathcal{V}_{1,3}$ (vide Figura 4.1), obtemos o esquema XML da VCN, mostrado na Figura 5.17. O esquema XML gerado possui um elemento raiz ($Traiz_{[V1,3]}$), que contém o elemento primário Materia obtido a partir da estrutura da VCN, e Materia possui um elemento para cada sub-elemento de Matéria na VCN $\mathcal{V}_{1,3}$.

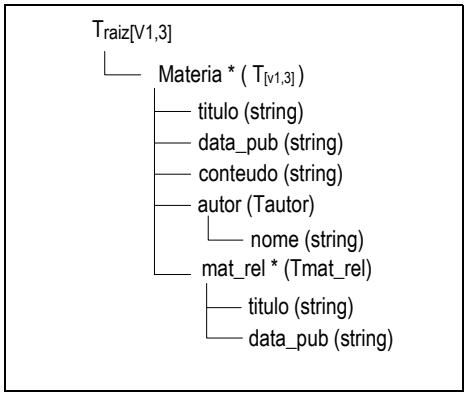


Figura 5.17: Esquema XML de $\mathcal{V}_{1,3}$

Passo 3: Gerar estilo XSLT

Neste passo, o documento de estilo da página XSQL é gerado a partir do esquema XML da VCN (da aplicação). Isso pode ser feito de forma automática, uma vez que as regras para renomear as *tags* do documento XML canônico podem ser inferidas a partir do esquema XML da VCN (da aplicação), como mostrado na Figura 5.18. Na seção 6.4, é apresentado o algoritmo GeraEstilo para geração dos documentos XSLT das páginas XSQL. Dado o esquema XML da VCN $\mathcal{V}_{1,3}$ da Figura 5.17, o esquema XML Canônico da VCN $\mathcal{V}_{1,3}$ é mapeado no esquema XML de $\mathcal{V}_{1,3}$ (da aplicação), como mostrado na Figura 5.18. A Figura 5.19 mostra um exemplo de documento XML canônico resultado do processamento da página XSQL da VCN $\mathcal{V}_{1,3}$ (Figura 5.16), e um documento XML no mesmo formato do esquema XML de $\mathcal{V}_{1,3}$.

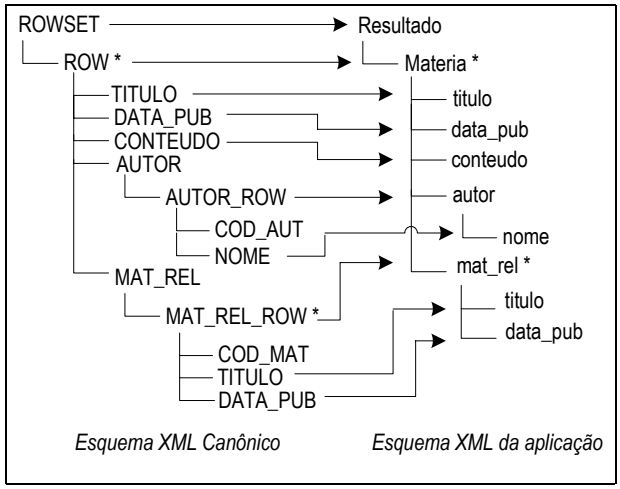


Figura 5.18: Esquemas XML de $\mathcal{V}_{1,3}$

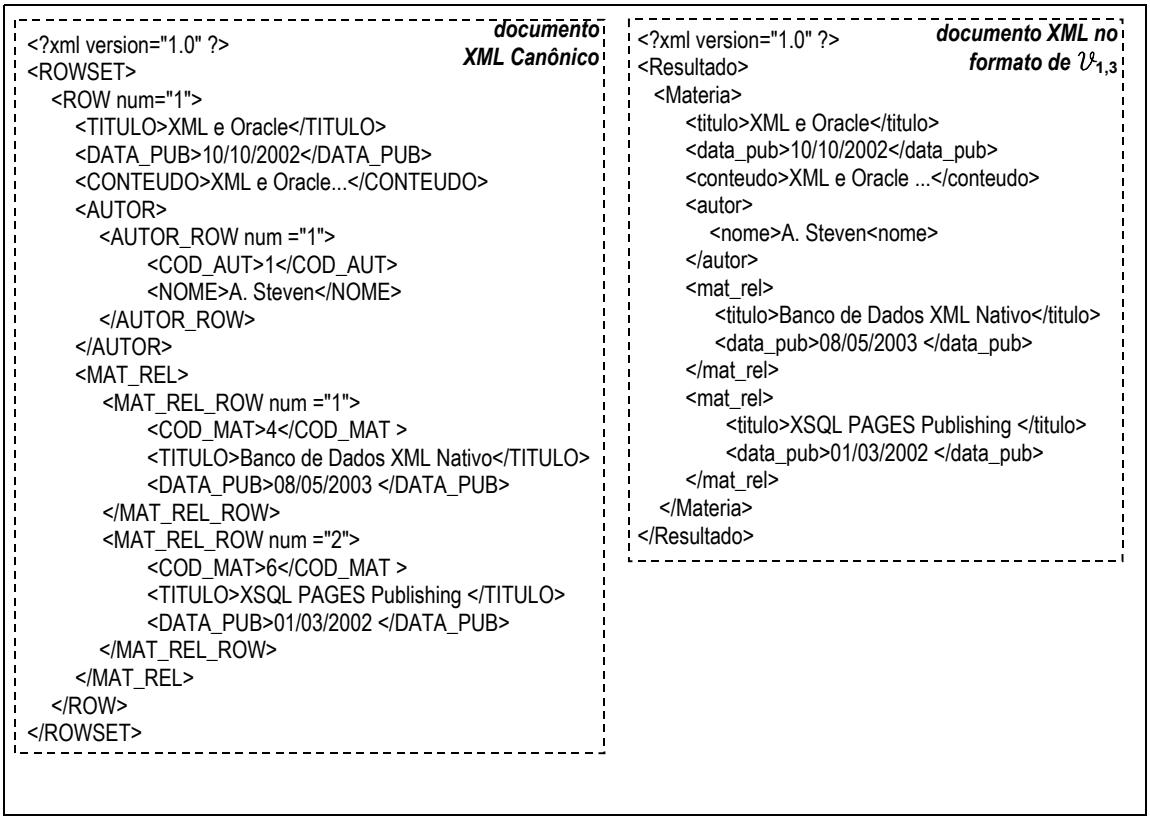


Figura 5.19: Exemplo de Documentos XML Canônico e no formato de $\mathcal{V}_{1,3}$

5.3 Manutenção das Visões de Contexto de Navegação

A atividade Manutenção das VCN's pode ser automatizada a partir das especificações conceituais das VCN's e das visões de objetos. A seguir, discutimos a manutenção das VCN's com relação a modificações no esquema do banco de dados e modificações no tipo de uma VCN:

- **Situação 1**: Modificações no esquema do banco de dados

Esquemas de banco de dados costumam sofrer modificações ao longo de sua existência. Essas modificações podem ocorrer devido a diversos fatores, tais como a necessidade de re-modelagem no banco, de forma a melhor atender aos requisitos de dados

dos usuários, a adição de VCN's, provocada por adição de funcionalidades à aplicação, e modificações na estrutura das VCN's.

Em relação a modificações no esquema do banco de dados, a manutenção das VCN's pode ser realizada de forma semi-automática, a partir da especificação das VCN's. Considere, por exemplo, o banco de dados Publicações_rel' da Figura 5.20, que corresponde ao esquema do banco de dados Publicações_rel modificado. Nesse novo esquema, as tabelas Pessoas_rel, Autores_rel e Leitores_rel representam tipos que pertencem a uma hierarquia de tipos. Observe que os atributos em comum das tabelas Autores_rel e Leitores_rel foram removidos das tabelas originais e adicionados a tabela Pessoas_rel.

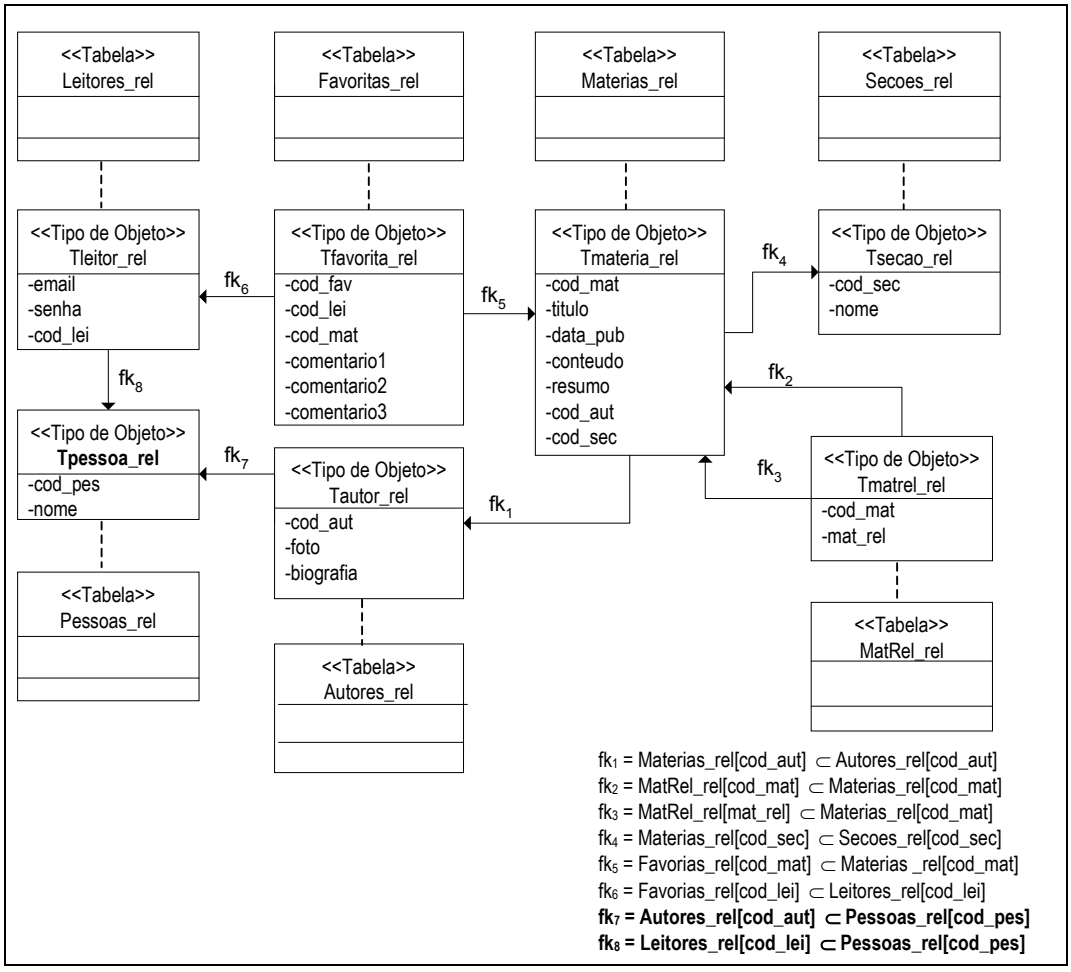


Figura 5.20: Esquema Relacional do Banco de Dados Publicações_rel'.

No caso em que as VCN's são definidas sobre o esquema das visões de objeto (Implementação em 3 camadas), somente as visões de objeto são afetadas, e, assim, precisam ser redefinidas. Nesse caso, o processo de manutenção das VCN's compreende dois passos:

1. Primeiramente, deve-se identificar as visões de objeto afetadas pelas modificações do esquema do banco de dados. Isso pode ser feito de modo automático, baseado nas modificações do esquema do banco de dados e nas assertivas de correspondência das visões de objeto. As AC's das visões de objeto afetadas são redefinidas de acordo com o novo esquema do banco de dados;

2. Para cada visão de objeto afetada, é gerada a consulta SQL que mapeia os dados do banco de dados em objetos da visão. Como discutido na seção 5.1.1, essa consulta é gerada automaticamente a partir das AC's da visão.

Considere o banco de dados Publicações_rel' da Figura 5.20. A partir dos cartões de especificação das visões de objeto do esquema Publicações (Figura 4.18) e das alterações do banco de dados Publicações_rel, é possível identificar (através das ACC's ψ_{12} e ψ_{13}) que a visão Autores_v foi afetada pela alteração. Neste caso, basta redefinir as AC's da visão de objeto de acordo com o novo esquema do banco de dados, gerando um novo cartão para a visão Autores_v (Figura 5.21), e em seguida, gerar a consulta SQL a partir desse cartão, mostrada na Figura 5.22.

Visão de Objeto: Autores_v	
Tabela Base:	(**ACE**: ψ_{10}: [Autores_v] ≡ [Autores_rel])
Autores_rel	(**ACO:** ψ_{11}: [T_{autor}, {cod_aut}] ≡ [T_{autor_rel}, {cod_aut}])
Atributos:	**ACC's:**
cod_aut: integer	ψ_{12}: [T_{autor}•cod_aut] ≡ [$T_{autores\_rel}$•cod_pes]
nome: string	ψ_{13}: [T_{autor}•nome] ≡ [$T_{autores\_rel}$•fk_7•nome]
foto: string	ψ_{14}: [T_{autor}•foto] ≡ [$T_{autores\_rel}$•foto]
biografia: string	ψ_{15}: [T_{autor}•biografia] ≡ [$T_{autores\_rel}$•biografia]

Figura 5.21: Cartão de especificação da visão de objeto Autores_v

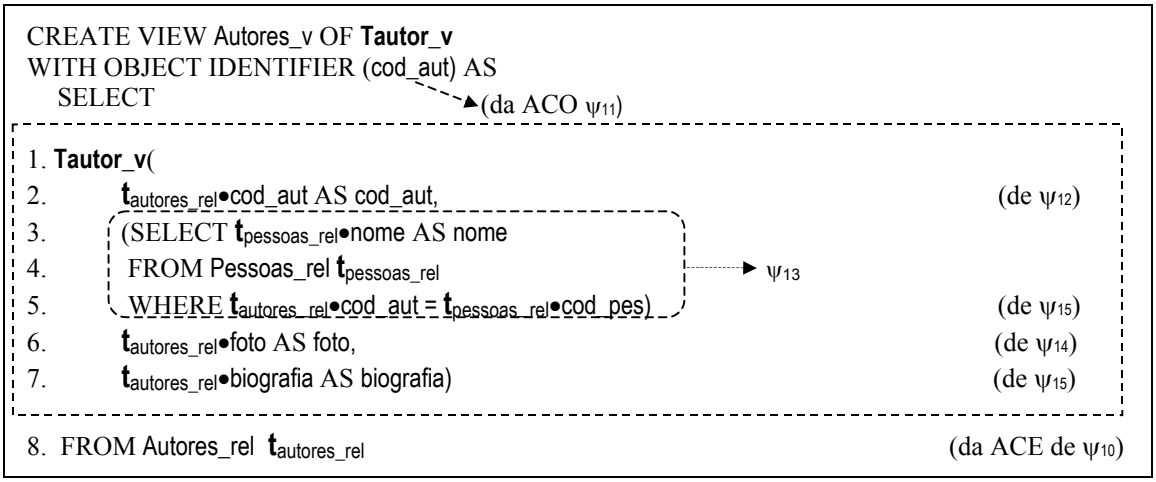


Figura 5.22: Definição da Visão de Objetos Autores_v

No caso em que as VCN's são definidas sobre esquema do banco de dados (Implementação em 2 camadas), todas as VCN's afetadas precisam ser redefinidas. O processo de manutenção consiste em três passos:

1. Primeiramente, deve-se identificar as visões de objeto afetadas pelas modificações do esquema do banco de dados e redefinir suas assertivas, conforme definido no passo 1 do caso anterior;

2. a partir das visões de objeto afetadas e de suas novas assertivas, pode-se então, automaticamente, identificar as VCN's afetadas e gerar as AC's da VCN com o novo esquema banco de dados, como discutido na seção 5.1.2. É importante notar que a especificação conceitual das VCN's não é afetada, uma vez que estas são especificadas sobre o esquema da visão de objeto;

3. para cada VCN afetada, é gerada a consulta SQL que define a VCN sobre o esquema do banco de dados. Como discutido na seção 5.1.2, a consulta SQL é gerada automaticamente, a partir das AC's da VCN com o novo esquema do banco de dados.

- **Situação 2**: Modificações do tipo de uma VCN

Em relação a modificações do tipo de uma VCN, devemos analisar os seguintes casos:

1. Se as modificações não afetam o tipo da visão de objeto base da VCN, a VCN é especificada novamente, utilizando a Ferramenta para Geração da Especificação das VCN's e um novo cartão de especificação da VCN é gerado. A VCN é então publicada de forma automática, de acordo com a abordagem adotada (como visão de objeto ou como visão XML), baseado no novo cartão.

2. Se as modificações afetam o tipo da visão de objetos base, esse tipo deve ser redefinido, considerando as modificações no tipo da VCN. O novo tipo é então integrado com os tipos das demais visões de objetos, de acordo com o apresentado na seção 4.2.1, e a visão de objeto base da VCN é especificada. O mesmo procedimento descrito no caso 1 é, então, executado.

Suponha, por exemplo, que a estrutura da VCN $\mathcal{V}_{1,3}$ (UID $\mathcal{U}_1$, na Figura 4.1) é alterada para Materia(titulo, data de publicação, Autor(nome, foto, biografia)). Nesse caso, o tipo da visão de objeto base (Materias_v) não é afetado, pois continua atendendo aos requisitos de dados da VCN $\mathcal{V}_{1,3}$. Essa VCN é especificada novamente e um novo cartão de especificação é gerado, mostrado na Figura 5.23. Suponha que a abordagem escolhida para implementação de $\mathcal{V}_{1,3}$ é implementação como visões de objetos em 3 camadas. A consulta

VCN: $\mathcal{V}_{1,3}$	
Parâmetros: *param*: integer	
Visão de Objeto Base: Materias_v	**Condição de Seleção** (**ACE**: $[\psi_1^{[V1,3]} : V_{1,3}] \equiv$ [Materias_v[cod_mat = *param*]]) cod_mat = *par*
Atributos:	**Assertivas de Correspondência**
titulo: string	$\psi_2^{[V1,3]}$: $[T_{[V1,3]} \bullet \text{titulo}] \equiv [T_{\text{materia}} \bullet \text{titulo}]$
data_de_publicação: string	$\psi_3^{[V1,3]}$: $[T_{[V1,3]} \bullet \text{data\_publicação}] \equiv [T_{\text{materia}} \bullet \text{data\_pub}]$
autor: <$T_{[Autor]}$>	$\psi_4^{[V1,3]}$: $[T_{[V1,3]} \bullet \text{autor}] \equiv [T_{\text{materia}} \bullet \text{autor}]$
nome: string	$\psi_5^{[V1,3]}$: $[T_{[autor]} \bullet \text{nome}] \equiv [T_{\text{autor}} \bullet \text{nome}]$
foto: string	$\psi_6^{[V1,3]}$: $[T_{[autor]} \bullet \text{foto}] \equiv [T_{\text{autor}} \bullet \text{foto}]$
biografia: string	$\psi_7^{[V1,3]}$: $[T_{[autor]} \bullet \text{biografia}] \equiv [T_{\text{autor}} \bullet \text{biografia}]$
Elos:	
Operações: imprimirMateria() enviarMateria()	
Ordenação:	

Figura 5.23: Cartão de especificação da VCN $\mathcal{V}_{1,3}$

SQL que define $\mathcal{V}_{1,3}$ é gerada automaticamente a partir das novas AC's da VCN com a visão Materias_v do esquema Publicações, especificadas no novo cartão de $\mathcal{V}_{1,3}$. A Figura 5.24 mostra a consulta que define $\mathcal{V}_{1,3}$ e as AC's da VCN usadas na geração da consulta.

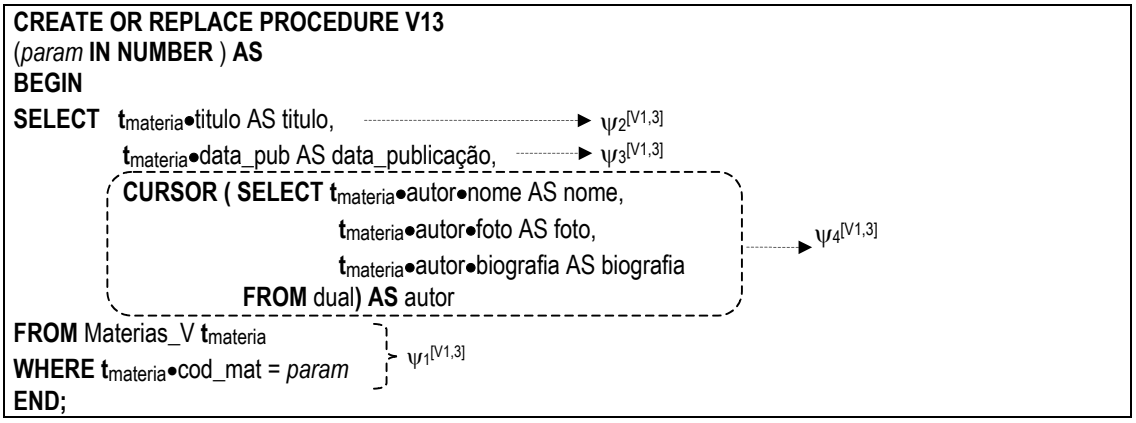


Figura 5.24: Definição da VCN $\mathcal{V}_{1,3}$ sobre esquema da visão de objeto Materias_v

Capítulo 6
Ferramentas e Algoritmos

Este capítulo está organizado como se segue. Na Seção 6.1, descrevemos as ferramentas do Ambiente proposto neste trabalho, denominado ProDIWA. Na Seção 6.2, apresentamos o algoritmo GeraConsultaSQL que gera, a partir da AC´s da VCN, a consultas SQL que define a VCN. Na Seção 6.3, apresentamos o algoritmo GeraAssertivas que gera, a partir das AC's da VCN com a visão de objeto base, as AC's da VCN com o esquema do banco de dados. Na Seção 6.3, apresentamos o algoritmo GeraEstilo, que gera os estilos XSLT, contendo as regras para transformação de documentos XML Canônicos no formato das VCN's.

6.1 Ferramentas do Ambiente Proposto

A Figura 6.1 mostra as ferramentas do ambiente proposto neste trabalho, as quais são descritas a seguir.

- **Ferramentas de Projeto**

  - **Ferramenta para Especificação das Visões de Objeto**

Esta ferramenta apóia a atividade de especificação das visões de objeto. A especificação de uma visão de objeto com a ferramenta é feita conforme descrito na seção 4.2.2.

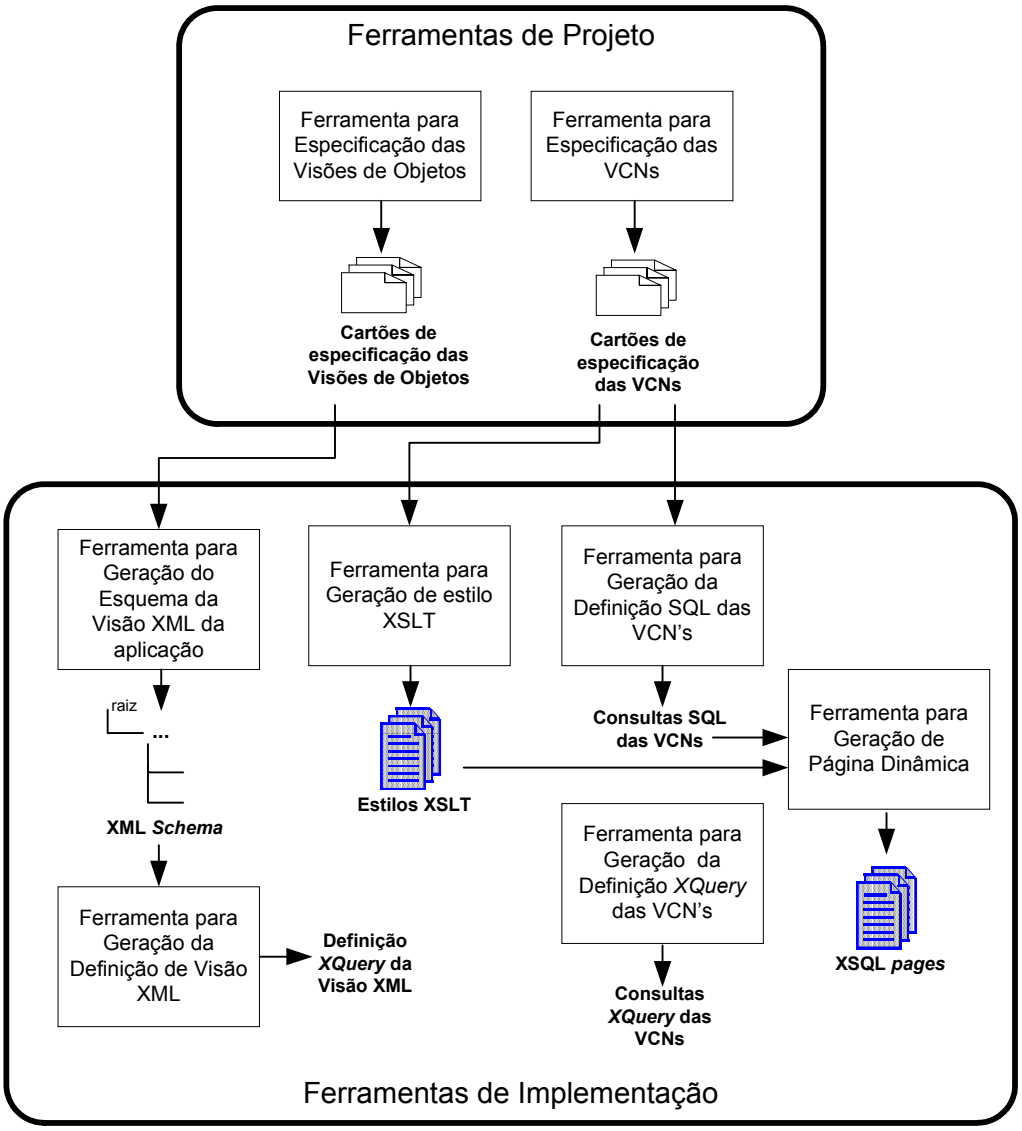


Figura 6.1: Ferramentas do ambiente ProDIWA

- **Ferramenta para Especificação das VCN´s**

Esta ferramenta apóia a atividade de especificação das VCN´s. A especificação de uma VCN com a ferramenta é feita conforme descrito na seção 4.3. No caso em que a VCN é definida sobre o esquema do banco de dados (implementação em 2 camadas), as assertivas de correspondência da VCN com o banco são geradas usando o algoritmo de GeraAssertivas, apresentado na seção 6.3.

- **Ferramentas de Implementação**

  - **Ferramenta para Geração da Definição SQL das VCN´s**

Esta ferramenta gera a definição SQL da VCN. Como vimos no Capítulo 5, essa consulta pode ser definida sobre o esquema da visão de objeto base ou sobre o esquema do banco de dados (relacional ou objteto-relacional). Na seção 6.2, apresentamos o algoritmo GeraConsultaSQL, o qual gera, a partir das AC's da VCN, a consulta SQL que define a VCN.

  - **Ferramenta para Geração de Página Dinâmica**

Esta ferramenta gera a página dinâmica contendo a consulta SQL que define a VCN. Nesse trabalho, as páginas geradas utilizam a tecnologia XSQL *Page*, e referenciam os estilos XSLT gerados para as VCN's, conforme discutido na seção 5.2.3. Essa ferramenta pode ser estendida para permitir a geração de páginas dinâmicas utilizando outras tecnologias.

  - **Ferramenta para Geração de Estilo XSLT**

Esta ferramenta gera um estilo XSLT o qual contém as regras para transformação do documento XML Canônico, conforme discutido 5.2.3. Na Seção 6.4 apresentamos o algoritmo GeraEstilo, o qual gera o estilo XSLT a partir do esquema XML da VCN.

  - **Ferramenta para Geração do Esquema da Visão XML da aplicação**

Esta ferramenta gera, a partir do esquema das visões de objeto da aplicação e suas assertivas de correspondências com o banco, o esquema XML da aplicação e suas AC's com o esquema da visão XML do banco de dados. A ferramenta é utilizada no caso da implementação das VCN's usando *middlewares* como o *Silkroute*, o *XPeranto* e o XML *Publisher*, conforme descrito nas seções 5.2.1 e 5.2.2.

  - **Ferramenta para Geração da Definição de Visão XML**

Esta ferramenta gera a consulta *XQuery* que define a visão XML da aplicação. Essa consulta é definida sobre o esquema da visão XML *Default* e é gerada a partir das AC's da

visão XML com a visão XML *Default*. A ferramenta é utilizada no caso da implementação das VCN's usando *middlewares* como o *Silkroute* e o *XPeranto*, conforme descrito na seção 5.2.1.

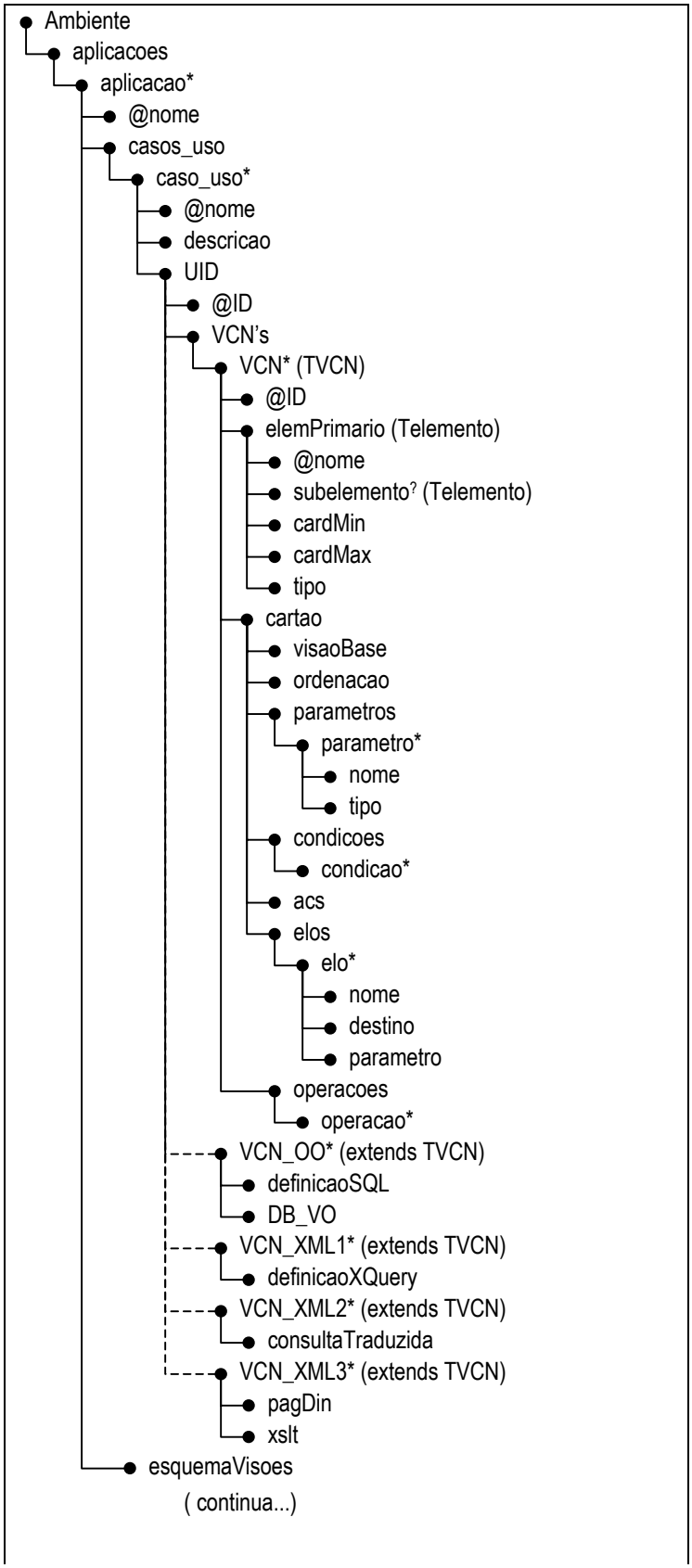
- **Ferramenta para Geração da Definição *XQuery* das VCN´s**

Esta ferramenta gera a consulta XQuery que define a VCN sobre o esquema XML da aplicação. A ferramenta é utilizada no caso da implementação das VCN's usando *middlewares* como o *Silkroute*, o *XPeranto* e o XML *Publisher*, conforme descrito nas seções 5.2.1 e 5.2.2.

- **Ferramenta para Geração das Visões de Objeto**

A ferramenta VBA (View by Assertion) proposta em [42], gera automaticamente a definição SQL das visões de objeto baseada na especificação da visão.

No ambiente ProDIWA, os objetos de uma dada aplicação (UID's, VCN's, visões de objeto, etc) são armazenado em um documento XML cujo esquema é mostrado na Figura 6.2 .


Ambiente
aplicacoes
aplicacao*
@nome
casos_uso
caso_uso*
@nome
descricao
UID
@ID
VCN's
VCN* (TVCN)
@ID
elemPrimario (Telemento)
@nome
subelemento? (Telemento)
cardMin
cardMax
tipo
cartao
visaoBase
ordenacao
parametros
parametro*
nome
tipo
condicoes
condicao*
acs
elos
elo*
nome
destino
parametro
operacoes
operacao*
VCN_OO* (extends TVCN)
definicaoSQL
DB_VO
VCN_XML1* (extends TVCN)
definicaoXQuery
VCN_XML2* (extends TVCN)
consultaTraduzida
VCN_XML3* (extends TVCN)
pagDin
xslt
esquemaVisoes
(continua...)

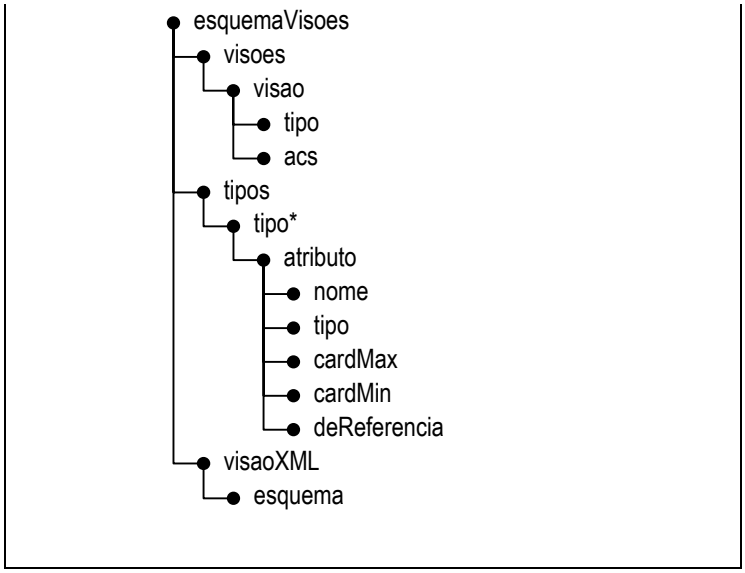


Figura 6.2: Esquema do Documento XML de armazenamento do ambiente ProDIWA

6.2 Algoritmo GeraConsultaSQL

Nesta seção, apresentamos o algoritmo GeraConsultaSQL, o qual gera a consulta SQL:1999 que realiza o mapeamento definido pelas assertivas da VCN. Essas assertivas definem a correspondência da VCN com a visão de objeto base ou com o banco de dados. O algoritmo GeraConsultaSQL é uma adaptação do algoritmo proposto em [42], o qual gera a consulta SQL que define uma visão de objeto a partir das AC's da visão de objeto. No resto desta seção, considere a VCN **V**, cujos objetos são do tipo $\mathbf{T}_{[V]}$.

Casos	Definição da Visão
1. A extensão de **V** é definida pela ACE [**V**] ≡ [$\mathbf{R}_b$], onde $\mathbf{R}_b$ é do tipo $\mathbf{T}_b$, e **o** é um critério de ordenação para os objetos da VCN, se for apresentado.	SELECT **Construtor_**$\mathbf{T_v}$**-**$\mathbf{T_b}$ ($\mathbf{t_b}$) FROM $\mathbf{R_b}$ $\mathbf{t_b}$ [ORDER BY **o**]
2. A extensão de **V** é definida pela ACE [**V**] ≡[$\mathbf{R_b}$[p]], onde $\mathbf{R_b}$ é do tipo $\mathbf{T_b}$, p é um predicado condicional e **o** é um critério de ordenação para os objetos da VCN, se for apresentado.	SELECT **Construtor_**$\mathbf{T_v}$**-**$\mathbf{T_b}$ ($\mathbf{t_b}$) FROM $\mathbf{R_b}$ $\mathbf{t_b}$ WHERE p($\mathbf{t_b}$) [ORDER BY **o**]

Tabela 6-1: Casos do algoritmo GeraConsultaSQL

A Tabela 6-1 mostra o padrão das consultas geradas para os dois tipos de ACE's. O construtor **Construtor_T_v-T_b** cria um objeto t_v do tipo T_v, a partir de um objeto (tupla) t_b da visão de objeto base ou tabela base R_b, onde T_b é o tipo de R_b, tal que $t_v \equiv t_b$. Os valores dos atributos do objeto t_v criado são definidos de acordo com as assertivas de correspondência de caminho de T_v & T_b. Se for definido um critério de ordenação **o** para os objetos da VCN **V** no seu cartão de especificação, a consulta gerada apresenta a cláusula ORDER BY. O construtor **Construtor_T_v-T_b** é gerado de forma automática, a partir das ACC's de T_v & T_b, pelos algoritmos **GeraConstrutorRelacional** e **GeraConstrutorObjetoRelacional**, que são adaptações dos algoritmos propostos em [42]. O algoritmo **GeraConstrutorRelacional**, apresentado na Seção 6.2.1, é utilizado quando as VCN's são especificadas sobre o banco de dados relacional. Já o algoritmo **GeraConstrutorObjetoRelacional**, apresentado na Seção 6.2.2, é utilizado quando as VCN's são especificadas sobre um esquema orientado a objetos, que pode ser o esquema de uma visão de objeto ou do banco de dados objeto-relacional.

É importante notar que outros tipos de consultas (união, intersecção e diferença), que têm mais de uma visão ou tabela base, podem ser tratadas da seguinte forma: no caso em que o tipo dos objetos das tabelas base são diferentes, primeiro, cria-se uma consulta para cada tabela base; assim são criados diferentes construtores para cada tabela base. Pode-se definir, então, uma consulta (de união, intersecção ou diferença) sobre as consultas criadas.

Exemplo 6.1

Considere o esquema do banco de dados **Publicações_rel** (Figura 4.19), e a VCN $\mathcal{V}_{1,3}$, cujo cartão de especificação é apresentado na Figura 5.8.

De acordo com o algoritmo GeraConsultaSQL, inicialmente a ACE $\delta_1^{[V]}$ da VCN é analisada. Como $\delta_1^{[V1,3]}$ é do tipo subconjunto, pois possui um predicado condicional que especifica a condição de seleção dos objetos da VCN, a definição da VCN será gerada pelo caso 2 da Tabela 6-1. A assertiva $\delta_1^{[V1,3]}$ especifica que para cada objeto $t_{[V1,3]}$ de $\mathcal{V}_{1,3}$, existe um objeto $t_{materias\_rel}$ na tabela base **Materias_rel** tal que $t_{materia} \equiv t_{materia\_rel}$. A cláusula FROM na definição da visão é composta pela tabela **Materias_rel**. A cláusula WHERE é composta

pelo predicado condicional p, onde p = [cod_mat = param]. A Figura 6.3 apresenta a definição da VCN $\mathcal{V}_{1,3}$ gerada pelo algoritmo.

O construtor **Construtor_**$T_{[V1,3]}$-$T_{materias\_rel}$ que aparece na cláusula SELECT da definição da visão gera um objeto do tipo $T_{[V1,3]}$ a partir de um objeto $t_{materias\_rel}$ do tipo $T_{materias\_rel}$, de forma que o objeto criado é semanticamente equivalente ao objeto $t_{materias\_rel}$. Para isto, os valores dos atributos do objeto criado são definidos de acordo com as AC's de $T_{[V1,3]}$ & $T_{materias\_rel}$, como veremos na Seção 6.2.1.

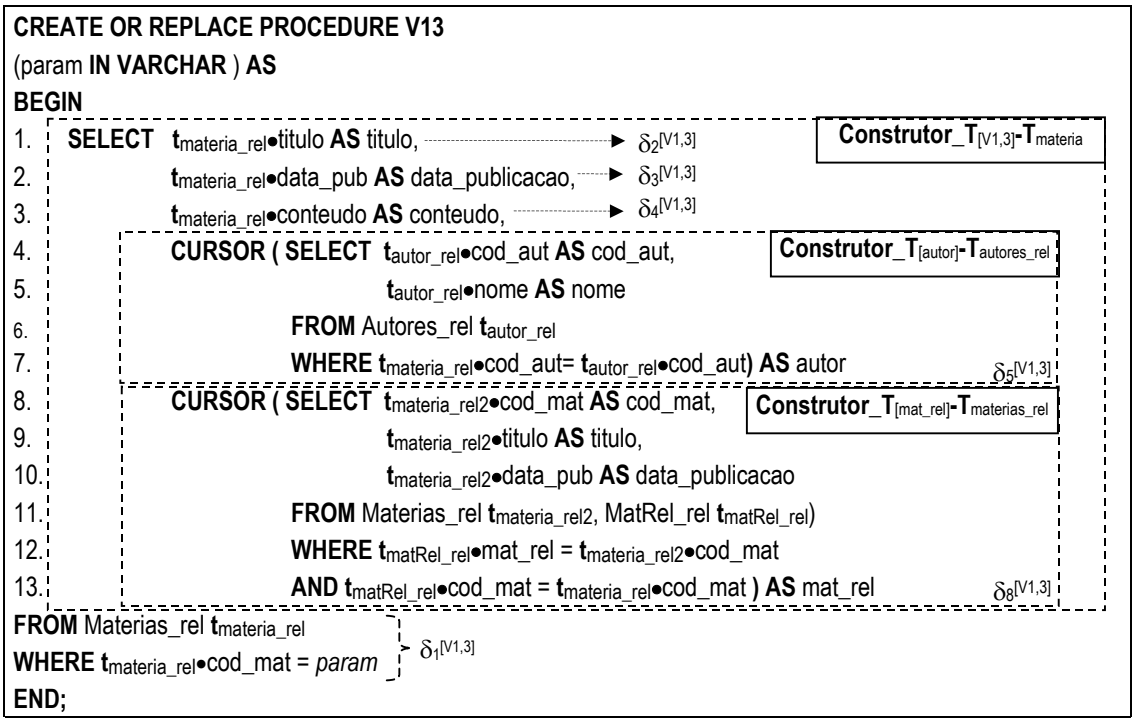


Figura 6.3: Definição da VCN $\mathcal{V}_{1,3}$ sobre esquema do banco de dados

6.2.1 Algoritmo GeraConstrutorRelacional

Nesta seção considere a VCN **V,** cujos objetos são do tipo $T_{v,}$ e R_b , a tabela ou visão pivô de **V** de tipo T_b. O algoritmo GeraConstrutorRelacional recebe como entrada o tipo T_v e o tipo base T_b e gera, com base nas ACC's de T_v & T_b, um construtor de objetos do tipo T_v, o qual cria uma instância t_v de T_v a partir de uma tupla t_b de T_b tal que t_v e t_b são semanticamente equivalentes ($t_v \equiv t_b$). Sejam C_1, ..., e C_m atributos de $T_{v,}$ então, o construtor

gerado tem a seguinte forma: $\mathbf{Q}_{C1}(t_b)$ AS C_1, ... , $\mathbf{Q}_{Cm}(t_b)$ AS C_m, onde $\mathbf{Q}_{Ci}(t_b)$, $1 \le i \le m$, é o código SQL que gera o valor do atributo C_i para um objeto da visão t_v a partir da tupla t_b. O alias de $\mathbf{Q}_{Ci}(t_b)$ (AS C_i) serve para renomear as propriedades dos objetos retornados pelo código SQL de acordo com os elementos da VCN. Os valores dos atributos de t_v são definidos a partir das ACC's de T_v & T_b, como mostrado na Tabela 6-2. Para essa Tabela, considere **CaminhoRel φ**, **Juncaoφ** e **Juncaoφ'** como definido a seguir:

Definição 5.8: (**CaminhoRel φ**) Considere $\varphi = l_1 \bullet l_2 \bullet ... \bullet l_n$ um caminho de T_b (Figura 6.4), onde:

(i) l_1 é uma ligação de chave estrangeira dada por $R_b[f_1^{l1}, ..., f_{m_1}^{l1}] \subseteq R_{l1}[g_1^{l1},.., g_{m_1}^{l1}]$ ou inversa da chave estrangeira dada por $R_{l1}[g_1^{l1}, ..., g_{m_1}^{l1}] \subseteq R_b[f_1^{l1}, ..., f_{m_1}^{l1}]$;

(ii) l_i, é uma ligação de chave estrangeira dada por $R_{l_{i-1}}[f_1^{l_i}, ..., f_{m_i}^{l_i}] \subseteq R_{l_i}[g_1^{l_i},.., g_{m_i}^{l_i}]$ ou inversa da chave estrangeira dada por $R_{l_i}[g_1^{l_i}, ..., g_{m_i}^{l_i}] \subseteq R_{l_{i-1}}[f_1^{l_i}, ..., f_{m_i}^{l_i}]$, $2 \le i \le n$;

Definição 5.9: (**Juncaoφ**) Considere $\varphi = l_1 \bullet l_2 \bullet ... \bullet l_n$ um caminho de T_b como definido na Figura 6.4.

Juncaoφ = $R_{l_1}\ t_{R_{l_1}}$,....., $R_{l_n}\ t_{R_{l_n}}$ WHERE $t_{R_b} \bullet f_k^{l1} = t_{R_{l_1}} \bullet g_k^{l1}$ AND $t_{R_{l_{j-1}}} \bullet f_w^{l_j} = t_{R_{l_j}} \bullet g_w^{l_j}$, onde $1 \le k \le m_1$, $1 \le w \le m_j$ e $2 \le j \le n$. Assim, dado uma instância t_b de T_b, temos que: $t_b \bullet \varphi$ = SELECT $t_{R_{l_n}}$ FROM **Juncaoφ .**

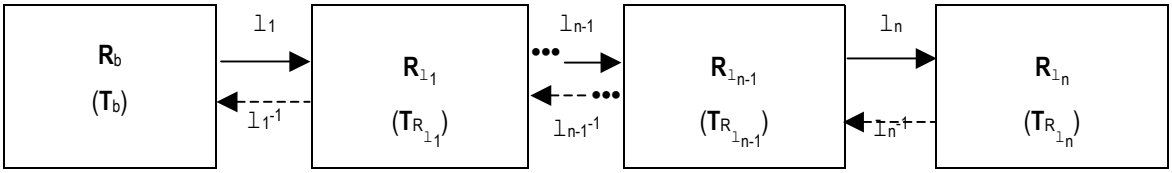


Figura 6.4: CaminhoRel φ

Definição 5.10: (**Juncaoφ'**) Considere $\varphi = l_1 \bullet l_2 \bullet ... \bullet l_n$ um caminho de T_b como definido na Figura 6.4 e p um predicado condicional.

Juncaoφ' = $\mathbf{R}_{l_1}$ $t_{R_{l_1}}$,….., $\mathbf{R}_{l_n}$ $t_{R_{l_n}}$ WHERE $t_{R_b} \bullet f_k^{l_1} = t_{R_{l_1}} \bullet g_k^{l_1}$ AND $t_{R_{l_{j-1}}} \bullet f_w^{l_j} = t_{R_{l_j}} \bullet g_w^{l_j}$ AND p ($t_{R_{l_n}}$) onde $1 \leq k \leq m_l$, $1 \leq w \leq m_j$ e $2 \leq j \leq n$. Assim, dado uma instância t_b de T_b, temos que: $t_b \bullet \varphi$ = SELECT $t_{R_{l_n}}$ FROM **Juncaoφ' .**

Casos	$Q_c(t_b)$
1. c é monovalorado de valor atômico e ψ_c:[$T_v \bullet$ **c**] ≡ [$T_b \bullet$ **a**], onde **a** é um atributo monovalorado atômico de T_b	$t_b \bullet$ **a**
2. c é monovalorado de valor atômico e ψ_c:[$T_v \bullet$ **c**] ≡ [$T_b \bullet \varphi \bullet$ **a**], onde φ é um caminho de T_b, **a** é um atributo monovalorado atômico e $t_{R_{l_n}}$ uma tupla da tabela R_{l_n}	(SELECT $t_{R_{l_n}} \bullet$ **a** FROM **Juncaoφ**)
3. c é monovalorado de valor estruturado de tipo T_c e ψ_c: [$T_v \bullet$ **c**] ≡ [$T_b \bullet \varphi$], onde φ é um caminho de T_b, $T_{R_{l_n}}$ o tipo de φ e $t_{R_{l_n}}$ uma tupla da tabela R_{l_n}.	CURSOR (SELECT **Construtor_**T_c-$T_{R_{l_n}}$($t_{R_{l_n}}$) FROM **Juncaoφ**)
4. c é multivalorado de valor atômico e ψ_c: [$T_v \bullet$ **c**] ≡ [$T_b \bullet \varphi \bullet$ **a**], onde φ é um caminho multivalorado de T_b, $T_{R_{l_n}}$ o tipo de φ, $t_{R_{l_n}}$ uma tupla da tabela R_{l_n}, e **a** um atributo monovalorado de valor atômico de $T_{R_{l_n}}$.	CURSOR (SELECT $t_{R_{l_n}} \bullet$ **a** FROM **Juncaoφ**)
5. c é multivalorado de valor atômico e ψ_c: [$T_v \bullet$ **c**] ≡ [$T_b \bullet \varphi \bullet$ **a**[p]], onde φ é um caminho multivalorado de T_b, $T_{R_{l_n}}$ o tipo de φ, $t_{R_{l_n}}$ uma tupla da tabela R_{l_n}, **a** um atributo monovalorado de valor atômico de $T_{R_{l_n}}$, p é um predicado condicional e **o** é um critério de ordenação, se for apresentado.	CURSOR (SELECT $t_{R_{l_n}} \bullet$ **a** FROM **Juncaoφ'** [ORDER BY **o**])
6. c é multivalorado de valor estruturado de tipo T_c e ψ_c: [$T_v \bullet$ **c**] ≡ [$T_b \bullet \varphi$], onde φ é um caminho de T_b, $T_{R_{l_n}}$ é o tipo de φ.	CURSOR (SELECT **Construtor_**T_c-$T_{R_{l_n}}$($t_{R_{l_n}}$) FROM **Junçãoφ**)
7. c é multivalorado de valor estruturado de tipo T_c e ψ_c: [$T_v \bullet$ **c**] ≡ [$T_b \bullet \varphi$[p]], onde φ é um caminho de T_b, $T_{R_{l_n}}$ é o tipo de φ, p é um predicado condicional e **o** é um critério de ordenação, se for apresentado.	CURSOR (SELECT **Construtor_**T_c-$T_{R_{l_n}}$($t_{R_{l_n}}$) FROM **Juncaoφ'** [ORDER BY **o**]))
8. c é monovalorado de valor estruturado e ψ_c:[$T_v \bullet$ **c**,{c_1,c_2,...,c_w}] ≡ [T_b,{d_1,d_2,..., d_w}]	CURSOR (SELECT $t_b \bullet d_1$ AS c_1, $t_b \bullet d_2$ AS c_2, ..., $t_b \bullet d_w$ AS c_w FROM DUAL)
9. c é multivalorado de valor atômico e ψ_c: [$T_v \bullet$ **c**] ≡ [T_b, {d_1,d_2,...,d_w}].	CURSOR (SELECT $t_b \bullet d_1$, $t_b \bullet d_2$,...,$t_b \bullet d_w$ FROM DUAL)
10. c é multivalorado de valor atômico e ψ_c: [$T_v \bullet$ **c**] ≡ [$T_b \bullet \varphi$,{d_1,d_2,...,d_w}], onde φ é um caminho monovalorado de T_b e $t_{R_{l_n}}$ uma tupla da tabela R_{l_n}	CURSOR(SELECT $t_{R_{l_n}} \bullet d_1$, $t_{R_{l_n}} \bullet d_2$,..., $t_{R_{l_n}} \bullet d_w$ FROM **Juncaoφ**)
11. c é multivalorado de valor atômico e ψ_c: [$T_v \bullet$ **c**] ≡ [$T_b \bullet \varphi$[p],{d_1,d_2,...,d_w}], onde φ é um caminho monovalorado de T_b, $t_{R_{l_n}}$ uma tupla da tabela R_{l_n}, p é um predicado condicional e **o** é um critério de ordenação, se for apresentado.	CURSOR(SELECT $t_{R_{l_n}} \bullet d_1$, $t_{R_{l_n}} \bullet d_2$,..., $t_{R_{l_n}} \bullet d_w$ FROM **Juncaoφ'** [ORDER BY **o**]))

Tabela 6-2: Casos do algoritmo GeraConstrutorRelacional.

Exemplo 6.2

Considere o esquema do banco de dados **Publicações_rel** (Figura 5.1), a especificação da VCN $\mathcal{V}_{1,3}$, cujo cartão de especificação é apresentado na Figura 5.8, e a definição dessa VCN sobre o esquema **Publicações_rel**, mostrada na Figura 6.3. A seguir, discutimos como são gerados os valores para cada atributo do tipo $\mathbf{T}_{[V1,3]}$ (**Construtor**_ $\mathbf{T}_{[V1,3]}$-$\mathbf{T}_{materia\_rel}$), a partir das AC's de $\mathbf{T}_{[V1,3]}$ & $\mathbf{T}_{materia\_rel}$. No resto desta seção, seja $\mathbf{t}_{[V1,3]}$ o objeto criado pelo construtor a partir do objeto base $\mathbf{t}_{materias\_rel}$.

- O atributo titulo (linha 1 – Figura 6.3) é um atributo monovalorado atômico definido pela AC $\delta_2^{[V1,3]}$:[$\mathbf{T}_{[V1,3]}$•titulo] ≡ [$\mathbf{T}_{materias\_rel}$•titulo]. Do Caso 1 da Tabela 6-2, temos que $\mathbf{t}_{[V1,3]}$•titulo = $\mathbf{t}_{materias\_rel}$•titulo.
- O atributo data_publicacao (linha 2 – Figura 6.3) é um atributo monovalorado atômico definido pela AC $\delta_3^{[V1,3]}$:[$\mathbf{T}_{[V1,3]}$•data_publicacao] ≡ [$\mathbf{T}_{materias\_rel}$•data_pub]. Do Caso 1 da Tabela 6-2, temos que $\mathbf{t}_{[V1,3]}$•data_publicacao = $\mathbf{t}_{materias\_rel}$•data_pub.
- O atributo conteudo (linha 3 – Figura 6.3) é um atributo monovalorado atômico definido pela AC $\delta_4^{[V1,3]}$:[$\mathbf{T}_{[V1,3]}$•conteudo] ≡ [$\mathbf{T}_{materias\_rel}$•conteudo]. Do Caso 1 da Tabela 6-2, temos que $\mathbf{t}_{[V1,3]}$•conteudo = $\mathbf{t}_{materias\_rel}$•conteudo.
- O atributo autor (linha 4 a 7 – Figura 6.3) é um atributo monovalorado de valor estruturado, de tipo $\mathbf{T}_{[autor]}$, definido pela AC $\delta_5^{[V1,3]}$:[$\mathbf{T}_{[V1,3]}$•autor] ≡ [$\mathbf{T}_{materias\_rel}$•$fk_1$], onde $\mathbf{T}_{[autor]}$ é o tipo de autor e $\mathbf{T}_{autores\_rel}$ é o tipo do caminho fk_1, satisfazendo, assim, o Caso 3 da Tabela 6-2. Dado que $\varphi = fk_1$, onde fk_1 é uma ligação de chave estrangeira dada por **Materias_rel**[cod_aut] ⊆ **Autores_rel**[cod_aut], temos:

  Juncaoφ = Autores_rel $\mathbf{t}_{autores\_rel}$ WHERE $\mathbf{t}_{materias\_rel}$•cod_aut= $\mathbf{t}_{autores\_rel}$•cod_aut).

  Assim, do Caso 3 da Tabela 6-2, temos que:

  $\mathbf{T}_{[V1,3]}$•autor = CURSOR(SELECT **Construtor**_$\mathbf{T}_{[autor]}$-$\mathbf{T}_{autores_rel}$($\mathbf{t}_{autores_rel}$)
  FROM Autores_rel $\mathbf{t}_{autores\_rel}$
  WHERE $\mathbf{t}_{materias\_rel}$•cod_aut= $\mathbf{t}_{autores\_rel}$•cod_aut).

  Os valores dos atributos no construtor **Construtor**_$\mathbf{T}_{[autor]}$-$\mathbf{T}_{autores_rel}$ são definidos pelas ACC's de $\mathbf{T}_{[autor]}$&$\mathbf{T}_{autores\_rel}$, como mostrado a seguir. No resto desta seção, considere $\mathbf{t}_{[autor]}$ o objeto criado pelo **Construtor**_$\mathbf{T}_{[autor]}$-$\mathbf{T}_{autores_rel}$ a partir do objeto base $\mathbf{t}_{autores\_rel}$.

- O atributo cod_aut (linha 4 – Figura 6.3) é um atributo monovalorado atômico definido pela AC $\delta_6^{[V1,3]}$:[$\mathbf{T}_{[autor]}$•cod_aut] ≡ [$\mathbf{T}_{autores\_rel}$•cod_aut]. Do Caso 1 da Tabela 6-2, temos que $\mathbf{t}_{[autor]}$•cod_aut = $\mathbf{t}_{autores\_rel}$•cod_aut.
- O atributo nome (linha 5 – Figura 6.3) é um atributo monovalorado atômico definido pela AC $\delta_7^{[V1,3]}$:[$\mathbf{T}_{[autor]}$•nome] ≡ [$\mathbf{T}_{autores\_rel}$•nome]. Do Caso 1 da Tabela 6-2, temos que $\mathbf{t}_{[autor]}$•nome= $\mathbf{t}_{autores\_rel}$•nome.

– O atributo mat_rel (linha 8 a 13 – Figura 6.3) é um atributo multivalorado de valor estruturado definido pela AC $\delta_8^{[V1,3]}$: [$\mathbf{T}_{[V1,3]}$•mat_rel] ≡ [$\mathbf{T}_{materias\_rel}$•$fk_2^{-1}$•$fk_3$], onde $\mathbf{T}_{[mat\_rel]}$ é o tipo de mat_rel, e $\mathbf{T}_{materias\_rel}$ é o tipo do caminho fk_2^{-1}•fk_3, satisfazendo, assim, o Caso 6 da Tabela 6-2. Dado que φ = fk_2^{-1}•fk_3, onde fk_2 é uma ligação de chave estrangeira dada por **MatRel_rel**[cod_mat] ⊆ **Materias_rel**[cod_mat] e fk_3 é uma ligação de chave estrangeira dada por **MatRel_rel**[mat_rel] ⊆ **Materias_rel**[cod_mat], temos:

Juncaoφ = MatRel_rel $\mathbf{t}_{matRel\_rel}$, Materias_rel $\mathbf{t}_{materias\_rel2}$ WHERE $\mathbf{t}_{matRel\_rel}$•mat_rel = $\mathbf{t}_{materias\_rel2}$•cod_mat AND $\mathbf{t}_{matRel\_rel}$•cod_mat = $\mathbf{t}_{materias\_rel}$•cod_mat.

Assim, do Caso 5 da Tabela 6-2, temos que:

$\mathbf{t}_{[V1,3]}$•mat_rel = CURSOR (SELECT **Construtor**_$\mathbf{T}_{[mat_rel]}$-$\mathbf{T}_{materias_rel}$($\mathbf{t}_{maRelt_rel}$)
FROM MatRel_rel $\mathbf{t}_{matRel\_rel}$, Materias_rel $\mathbf{t}_{materias\_rel2}$
WHERE $\mathbf{t}_{matRel\_rel}$•mat_rel = $\mathbf{t}_{materias\_rel2}$•cod_mat
AND $\mathbf{t}_{matRel\_rel}$•cod_mat = $\mathbf{t}_{materias\_rel}$•cod_mat).

Os valores dos atributos no construtor **Construtor**_$\mathbf{T}_{[mat_rel]}$-$\mathbf{T}_{materias_rel}$ são definidos pelas ACC's de $\mathbf{T}_{[mat\_rel]}$-&$\mathbf{T}_{materias\_rel}$, como mostrado a seguir. No resto desta seção, considere $\mathbf{t}_{[mat\_rel]}$ o objeto criado pelo **Construtor**_$\mathbf{T}_{[mat_rel]}$-$\mathbf{T}_{materias_rel}$ a partir do objeto base $\mathbf{t}_{materias\_rel2}$.

- O atributo cod_mat (linha 8 – Figura 6.3) é um atributo monovalorado atômico definido pela AC $\delta_9^{[V1,3]}$:[$\mathbf{T}_{[mat\_rel]}$•cod_mat] ≡ [$\mathbf{T}_{materias\_rel}$•cod_mat]. Do Caso 1 da Tabela 6-2, temos que $\mathbf{t}_{mat\_rel}$•cod_mat = $\mathbf{t}_{materias\_rel}$•cod_mat.
- O atributo titulo (linha 9 – Figura 6.3) é um atributo monovalorado atômico definido pela AC $\delta_{10}^{[V1,3]}$:[$\mathbf{T}_{[mat\_rel]}$•titulo] ≡ [$\mathbf{T}_{materias\_rel}$•titulo]. Do Caso 1 da Tabela 6-2, temos que $\mathbf{t}_{mat\_rel}$•titulo = $\mathbf{t}_{materias\_rel2}$•titulo.

- O atributo data_publicacao (linha 10 – Figura 6.3) é um atributo monovalorado atômico definido pela AC $\delta_{11}^{[V1,3]}$:[$\mathbf{T}_{[mat\_rel]}$•data_publicacao] ≡ [$\mathbf{T}_{materias\_rel}$•data_pub]. Do Caso 1 da Tabela 6-2, temos que: $\mathbf{t}_{mat\_rel}$•data_publicacao = $\mathbf{t}_{materias\_rel2}$•data_pub.

6.2.2 Algoritmo GeraConstrutorObjetoRelacional

Nesta seção considere a VCN **V,** cujos objetos são do tipo $\mathbf{T}_v$, e $\mathbf{R}_b$, a tabela ou visão de objeto pivô de **V** de tipo $\mathbf{T}_b$. O algoritmo GeraConstrutorObjetoRelacional recebe como entrada o tipo $\mathbf{T}_v$ e o tipo base $\mathbf{T}_b$ e gera, com base nas ACC's de $\mathbf{T}_v$ & $\mathbf{T}_b$, um construtor de objetos do tipo $\mathbf{T}_v$, o qual cria uma instância $\mathbf{t}_v$ de $\mathbf{T}_v$ a partir de uma instância $\mathbf{t}_b$ de $\mathbf{T}_b$, tal que $\mathbf{t}_v$ e $\mathbf{t}_b$ são semanticamente equivalentes ($\mathbf{t}_v \equiv \mathbf{t}_b$). Sejam $\mathbf{C}_1$, ..., e $\mathbf{C}_m$ atributos de $\mathbf{T}_v$, então, o construtor gerado tem a seguinte forma: $\mathbf{Q}_{C1}(\mathbf{t}_b)$ AS $\mathbf{C}_1$, ... , $\mathbf{Q}_{Cm}(\mathbf{t}_b)$ AS $\mathbf{C}_M$, onde $\mathbf{Q}_{Ci}(\mathbf{t}_b)$, $1 \leq i \leq m$, é o código SQL:1999 que gera o valor do atributo $\mathbf{C}_i$ para um objeto da visão $\mathbf{t}_v$ a partir do objeto $\mathbf{t}_b$. O alias de $\mathbf{Q}_{Ci}(\mathbf{t}_b)$ (AS $\mathbf{C}_i$) serve para renomear as propriedades dos objetos retornados pelo código SQL gerado de acordo com os elementos da VCN. Os valores dos atributos de $\mathbf{t}_{[V]}$ são definidos a partir das ACC's de $\mathbf{T}_v$ & $\mathbf{T}_b$, como mostrado na Tabela 6-3. Os valores de $\mathbf{S}_\varphi$, $\mathbf{F}_\varphi$, e $\mathbf{W}_\varphi$ que aparecem nos casos 3, 5, 6, 7 e 8 da Tabela 6-3, são computados pelo procedimento GeraConsulta, apresentado na Figura 6.5. O procedimento GeraConsulta recebe como entrada um caminho φ de $\mathbf{T}_b$ e uma expressão de caminho **E**, e retorna uma tripla < $\mathbf{S}_\varphi$, $\mathbf{F}_\varphi$, $\mathbf{W}_\varphi$ >, de forma que a consulta dada por:

Select $\mathbf{S}_\varphi$ From $\mathbf{F}_\varphi$ Where $\mathbf{W}_\varphi$,

retorna os objetos em **E**•φ (**Select $\mathbf{S}_\varphi$ From $\mathbf{F}_\varphi$ Where $\mathbf{W}_\varphi$** = { $\mathbf{t}_\varphi$ | $\mathbf{t}_\varphi \in$ **E**•φ}). É importante notar que isso só é necessário para os casos em que o caminho φ é virtual ou multivalorado. No caso em que o caminho φ é direto, $\mathbf{W}_\varphi$ tem valor nulo, então **E**•φ = **Select $\mathbf{S}_\varphi$ From $\mathbf{F}_\varphi$**.

Casos	$\mathbf{Q_C(t_b)}$
1. c é monovalorado de valor atômico e ψ_c:[$\mathbf{T_v \bullet c}$]≡[$\mathbf{T_b \bullet a}$], onde **a** é um atributo monovalorado de valor atômico de $\mathbf{T_b}$	$\mathbf{t_b \bullet a}$
2. c é monovalorado de valor atômico e ψ_c:[$\mathbf{T_v \bullet c}$]≡[$\mathbf{T_b \bullet \varphi \bullet a}$], onde φ é um caminho direto de $\mathbf{T_b}$, $\mathbf{T_\varphi}$ o tipo de φ e **a** é um atributo monovalorado de valor atômico de $\mathbf{T_\varphi}$	$\mathbf{t_b \bullet \varphi \bullet a}$
3. c é monovalorado de valor atômico e ψ_c:[$\mathbf{T_v \bullet c}$]≡[$\mathbf{T_b \bullet \varphi \bullet a}$], onde φ é um caminho virtual de $\mathbf{T_b}$, $\mathbf{T_\varphi}$ o tipo de φ e **a** é um atributo monovalorado de valor atômico de $\mathbf{T_\varphi}$	(SELECT $\mathbf{S_\varphi \bullet a}$ FROM $\mathbf{F_\varphi}$ WHERE $\mathbf{W_\varphi}$)
4. c é monovalorado de valor estruturado e ψ_c:[$\mathbf{T_v \bullet c}$]≡[$\mathbf{T_b \bullet \varphi}$], onde φ é um caminho direto de $\mathbf{T_b}$, $\mathbf{T_\varphi}$ o tipo de φ e $\mathbf{T_c}$ é o tipo do atributo **c**	CURSOR (SELECT **Construtor_T_c-T_φ** ($\mathbf{t_b \bullet \varphi}$) FROM DUAL)
5. c é monovalorado de valor estruturado e ψ_c:[$\mathbf{T_v \bullet c}$]≡[$\mathbf{T_b \bullet \varphi}$], onde φ é um caminho virtual de $\mathbf{T_b}$, $\mathbf{T_\varphi}$ o tipo de φ e $\mathbf{T_c}$ é o tipo do atributo **c**	CURSOR (SELECT **Construtor_T_c-T_φ** ($\mathbf{S_\varphi}$) FROM $\mathbf{F_\varphi}$ WHERE $\mathbf{W_\varphi}$)
6. c é multivalorado de valor atômico e ψ_c:[$\mathbf{T_v \bullet c}$]≡[$\mathbf{T_b \bullet \varphi \bullet a}$], onde φ é um caminho de $\mathbf{T_b}$, $\mathbf{T_\varphi}$ o tipo de φ, **a** é um atributo monovalorado de valor atômico de $\mathbf{T_\varphi}$ e $\mathbf{T_c}$ é o tipo do atributo **c**	CURSOR (SELECT $\mathbf{S_\varphi \bullet a}$ FROM $\mathbf{F_\varphi}$ WHERE $\mathbf{W_\varphi}$)
7. c é multivalorado de valor atômico e ψ_c:[$\mathbf{T_v \bullet c}$]≡[$\mathbf{T_b \bullet \varphi \bullet a}$], onde φ é um caminho de $\mathbf{T_b}$, $\mathbf{T_\varphi}$ o tipo de φ, **a** é um atributo monovalorado de valor atômico de $\mathbf{T_\varphi}$, $\mathbf{T_c}$ é o tipo do atributo **c,** p é um predicado condicional e **o** é um critério de ordenação, se for apresentado.	CURSOR (SELECT $\mathbf{S_\varphi \bullet a}$ FROM $\mathbf{F_\varphi}$ WHERE $\mathbf{W_\varphi}$ AND p($\mathbf{t_b}$) [ORDER BY **o**])
8. c é multivalorado de valor atômico e ψ_c:[$\mathbf{T_v \bullet c}$]≡[$\mathbf{T_b \bullet \varphi}$], onde φ é um caminho de $\mathbf{T_b}$, $\mathbf{T_\varphi}$ o tipo de φ e $\mathbf{T_c}$ é o tipo do atributo **c**.	CURSOR (SELECT $\mathbf{S_\varphi}$•COLUMN_VALUE FROM $\mathbf{F_\varphi}$ WHERE $\mathbf{W_\varphi}$)
9. c é multivalorado de valor atômico e ψ_c:[$\mathbf{T_v \bullet c}$]≡[$\mathbf{T_b \bullet \varphi}$], onde φ é um caminho de $\mathbf{T_b}$, $\mathbf{T_\varphi}$ o tipo de φ, $\mathbf{T_c}$ é o tipo do atributo **c**, p é um predicado condicional e **o** é um critério de ordenação, se for apresentado.	CURSOR (SELECT $\mathbf{S_\varphi}$•COLUMN_VALUE FROM $\mathbf{F_\varphi}$ WHERE $\mathbf{W_\varphi}$ AND p($\mathbf{t_b}$) [ORDER BY **o**])
10. c é multivalorado de valor estruturado e ψ_c:[$\mathbf{T_v \bullet c}$]≡[$\mathbf{T_b \bullet \varphi}$], onde φ é um caminho de $\mathbf{T_b}$, $\mathbf{T_\varphi}$ o tipo de φ e $\mathbf{T_c}$ é o tipo dos objetos de **c**	CURSOR (SELECT **Construtor_T_c-T_φ** ($\mathbf{S_\varphi}$) FROM $\mathbf{F_\varphi}$ WHERE $\mathbf{W_\varphi}$)
11. c é multivalorado de valor estruturado e ψ_c:[$\mathbf{T_v \bullet c}$]≡[$\mathbf{T_b \bullet \varphi}$], onde φ é um caminho de $\mathbf{T_b}$, $\mathbf{T_\varphi}$ o tipo de φ, $\mathbf{T_c}$ é o tipo dos objetos de **c**, p é um predicado condicional e **o** é um critério de ordenação, se for apresentado.	CURSOR (SELECT **Construtor_T_c-T_φ** ($\mathbf{S_\varphi}$) FROM $\mathbf{F_\varphi}$ WHERE $\mathbf{W_\varphi}$ AND p($\mathbf{t_b}$) [ORDER BY **o**])

Tabela 6-3: Casos do algoritmo GeraConstrutorObjetoRelacional

Exemplo 6.3

Considere o esquema da visão de objeto **Materias_v** (vide Figura 4.18), a VCN $\mathcal{V}_{1,3}$, cujo cartão de especificação é apresentado na Figura 5.6, e as AC's de $\mathcal{V}_{1,3}$ com o esquema da visão **Materias_v** (vide Figura 5.6). O construtor **Construtor_$T_{[V1,3]}$-$T_{materia}$** gerado pelo algoritmo é mostrado na Figura 6.6. A seguir, mostramos como foi gerado o código

SQL:1999 para cada atributo de $\mathbf{T}_{[V1,3]}$. No resto desta seção, seja $\mathbf{t}_{[V1,3]}$ o objeto criado por esse construtor a partir do objeto base $\mathbf{t}_{materia}$.

GeraConsulta (φ : caminho de $\mathbf{T}_b$, **E**: uma expressão de caminho)
Var
  $\mathbf{S}_\varphi$: expressão de caminho
  $\mathbf{F}_\varphi$: lista de tabelas ou *nested tables*
  $\mathbf{W}_\varphi$: conjunções
Início
  $\mathbf{S}_\varphi$:= **E**
  Para cada ligação l_i de φ, 1≤ i ≤ n, **Faça**
    Caso 1: l_i é direta
      Caso 1.1: l_i é monovalorada
        $\mathbf{S}_\varphi$+= "•" + $\mathtt{l}_i$
      Caso 1.2: l_i é multivalorada
        $\mathbf{F}_\varphi$ += "TABLE(" + $\mathbf{S}_\varphi$+ "•" + $\mathtt{l}_i$ + ")" + $\mathbf{t}_{\mathtt{l}_i}$
        Se l_i é de referência **Então**
          $\mathbf{S}_\varphi$:= $\mathbf{t}_{\mathtt{l}_i}$ + "•COLUMN_VALUE"
        Senão
          $\mathbf{S}_\varphi$:= $\mathbf{t}_{\mathtt{l}_i}$
        Fim Se
    Caso 2: $\mathtt{l}_i$ é virtual
      Seja **R** a tabela de objetos (ou visão) referenciada por $\mathtt{l}_i$
      $\mathbf{F}_\varphi$ += **R** $\mathbf{t}_{\mathtt{l}_i}$
      Caso 2.1: $\mathtt{l}_i^{-1}$é monovalorada
        Se $\mathbf{S}_\varphi$contém um caminho de referência **Então**
          $\mathbf{W}_\varphi$ += $\mathbf{t}_{\mathtt{l}_i}$ + "•" + $\mathtt{l}_i^{-1}$ + "=" + $\mathbf{S}_\varphi$
        Senão
          $\mathbf{W}_\varphi$ += $\mathbf{t}_{\mathtt{l}_i}$ + "•" + $\mathtt{l}_i^{-1}$ + "= REF(" + $\mathbf{S}_\varphi$+ ")"
        Fim Se
        $\mathbf{S}_\varphi$:= $\mathbf{t}_{\mathtt{l}_i}$
      Caso 2.2: $\mathtt{l}_i^{-1}$ é multivalorada
        Se $\mathbf{S}_\varphi$contém um caminho de referência **Então**
          $\mathbf{W}_\varphi$ += $\mathbf{S}_\varphi$+ "IN " + "(SELECT * FROM TABLE (" + $\mathbf{t}_{\mathtt{l}_i}$ + "•" + $\mathtt{l}_i^{-1}$ + "))"
        Senão
          $\mathbf{W}_\varphi$ += "REF(" + $\mathbf{S}_\varphi$+ ") IN " +
                                "(SELECT * FROM TABLE (" + $\mathbf{t}_{\mathtt{l}_i}$ + "•" + $\mathtt{l}_i^{-1}$ + "))"
        Fim Se
        $\mathbf{S}_\varphi$:= $\mathbf{t}_{\mathtt{l}_i}$

  Fim Para
  Retorne ⟨ $\mathbf{S}_\varphi$, $\mathbf{F}_\varphi$, $\mathbf{W}_\varphi$ ⟩
Fim

Figura 6.5: Procedimento GeraConsulta

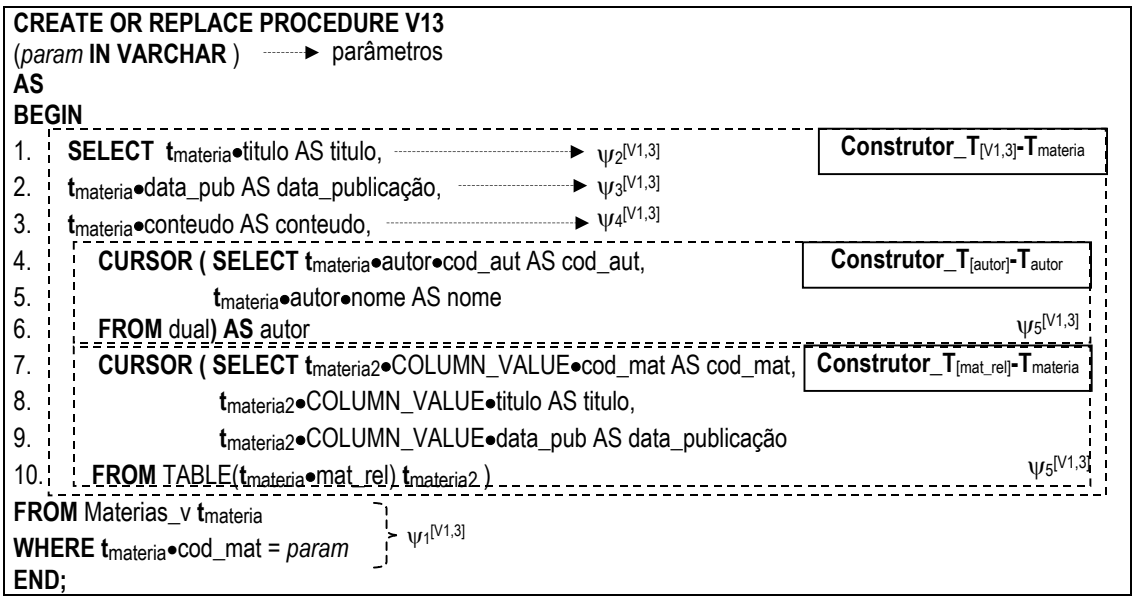


Figura 6.6: Definição da VCN $\mathcal{V}_{1,3}$ sobre visão de objeto Materias_v

- O atributo titulo (linha 1 - Figura 6.6) é um atributo monovalorado de valor atômico definido pelas AC $\psi_2^{[V1,3]}$: [$\mathbf{T}_{[V1,3]}$•**titulo**] ≡ [$\mathbf{T}_{materia}$•**titulo**]. Do Caso 1 da Tabela 6-3, temos que $\mathbf{t}_{[V1,3]}$•titulo = $\mathbf{t}_{materia}$•titulo.
- O atributo data_pub (linha 2 - Figura 6.6) é um atributo monovalorado de valor atômico definido pela AC $\psi_3^{[V1,3]}$: [$\mathbf{T}_{[V1,3]}$•**data_pub**] ≡ [$\mathbf{T}_{materia}$•**data_pub**]. Do Caso 1 da Tabela 6-3, temos que $\mathbf{t}_{[V1,3]}$•data_pub = $\mathbf{t}_{materia}$•data_pub.
- O atributo conteudo (linha 3 - Figura 6.6) é um atributo monovalorado de valor atômico definido pela AC $\psi_4^{[V1,3]}$: [$\mathbf{T}_{[V1,3]}$•**conteudo**] ≡ [$\mathbf{T}_{materia}$•**conteudo**]. Do Caso 1 da Tabela 6-3, temos que $\mathbf{t}_{[V1,3]}$•conteudo= $\mathbf{t}_{materia}$•conteudo.
- O atributo autor (linha 4 a 6 - Figura 6.6) é um atributo monovalorado de valor estruturado definido pela AC $\psi_5^{[V1,3]}$: [$\mathbf{T}_{[V1,3]}$•**autor**] ≡ [$\mathbf{T}_{materia}$•**autor**]. Do Caso 4 da Tabela 6-3, temos que:

$$\mathbf{t}_{[V1,3]}\text{•autor} = \text{CURSOR(SELECT Construtor\_}\mathbf{T}_{[autor]}\text{-}\mathbf{T}_{autor}(\mathbf{t}_{materia}\text{•autor})$$
$$\text{FROM DUAL)}$$

Os valores dos atributos do construtor **Construtor**_$\mathbf{T}_{[autor]}$-$\mathbf{T}_{autor}$ são definidos pelas ACC's de $\mathbf{T}_{[autor]}$&$\mathbf{T}_{autor}$, como mostrado a seguir.

- O atributo cod_aut (linha 4 - Figura 6.6) é monovalorado de valor atômico definido pela ACC $\psi_6^{[V1,3]}$:[$\mathbf{T}_{[autor]}$•**cod_aut**] ≡ [$\mathbf{T}_{autor}$•**cod_aut**]. Do Caso 2 da Tabela 6-3, temos que: $\mathbf{t}_{[autor]}$•cod_aut = $\mathbf{t}_{materia}$•autor•cod_aut.
- O atributo titulo (linha 5 - Figura 6.6) é monovalorado de valor atômico definido pela ACC $\psi_7^{[V1,3]}$:[$\mathbf{T}_{[autor]}$•**nome]** ≡ [$\mathbf{T}_{autor}$•**nome**]. Do Caso 2 da Tabela 6-3, temos que: $\mathbf{t}_{[autor]}$•nome = $\mathbf{t}_{materia}$•autor•nome.

– O atributo mat_rel (linhas 7 a 10 - Figura 6.6) é um atributo multivalorado de valor estruturado definido pela AC $\psi_8^{[V1,3]}$:[$\mathbf{T}_{[V1,3]}$•**mat_rel]** ≡ [$\mathbf{T}_{materia}$•**mat_rel**], onde $\mathbf{T}_{[mat\_rel]}$ é o tipo do atributo mat_rel e $\mathbf{T}_{materia}$ é o tipo do caminho mat_rel, satisfazendo, assim, o Caso 10 da Tabela 6-3. Dado que φ=mat_rel é um caminho multivalorado, GeraConsulta(φ, $\mathbf{t}_{materia}$) retorna: $\mathbf{S}_{\varphi}$="$\mathbf{t}_{materia2}$•COLUMN_VALUE"[4], $\mathbf{F}_{\varphi}$="TABLE($\mathbf{t}_{materia}$•**mat_rel**) $\mathbf{t}_{materia2}$" e $\mathbf{W}_{\varphi}$ tem valor nulo. Assim, do Caso 8 da Tabela 6-3 temos que:

$\mathbf{T}_{materia\_v}$•mat_rel=

```
CURSOR(
    SELECT Construtor_T_mat_rel–T_materia (t_materia2•COLUMN_VALUE)
    FROM TABLE( t_materia•mat_rel) t_materia2 )
```

Os valores dos atributos do construtor **Construtor_**$\mathbf{T}_{[mat\_rel]}$**-**$\mathbf{T}_{materia}$ são definidos pelas ACC's de $\mathbf{T}_{[mat\_rel]}$**&**$\mathbf{T}_{materia}$, como mostrado a seguir. No resto desta seção, considere $\mathbf{t}_{fmateria2}$ o objeto criado pelo construtor a partir do objeto base em $\mathbf{t}_{materia}$•COLUMN_VALUE.

- O atributo cod_mat (linha 7 - Figura 6.6) é monovalorado de valor atômico definido pela ACC $\psi_9^{[V1,3]}$:[$\mathbf{T}_{[mat\_rel]}$•**cod_mat**] ≡ [$\mathbf{T}_{materia}$•**cod_mat**]. Do Caso 1 da Tabela 6-3, temos que: $\mathbf{t}_{[mat\_rel]}$•cod_mat = $\mathbf{t}_{materia2}$•COLUMN_VALUE•cod_mat.
- O atributo titulo (linha 8 - Figura 6.6) é monovalorado de valor atômico definido pela ACC $\psi_{10}^{[V1,3]}$:[$\mathbf{T}_{[mat\_rel]}$•**titulo]** ≡ [$\mathbf{T}_{materia}$•**titulo**]. Do Caso 1 da Tabela 6-3, temos que: $\mathbf{t}_{[mat\_rel]}$•nome = $\mathbf{t}_{materia2}$•COLUMN_VALUE•nome.

[4] No Oracle 9i, usamos a expressão COLUMN_VALUE para denotar os elementos de uma nested table, quando seu tipo é atômico

- O atributo data_pub (linhas 9 - Figura 6.6) é um atributo monovalorado de valor estruturado definido pela ACC $\psi_{11}^{[V1,3]}$:[$\mathbf{T}_{[mat\_rel]}$•**data_pub**] ≡ [$\mathbf{T}_{materia}$•**data_pub**. Do Caso 1 da Tabela 6-3, temos que: $\mathbf{t}_{[mat\_rel]}$•data_pub = $\mathbf{t}_{materia2}$•COLUMN_VALUE•data_pub.

Exemplo 6.4

Considere o esquema do banco de dados Publicações_or apresentado na Figura 6.7 e a VCN $\mathcal{V}$, cujo cartão de especificação é mostrado na Figura 6.8, onde as AC's de $\mathcal{V}$ são especificadas sobre o esquema do banco de dados Publicações_or. O construtor **Construtor_**$\mathbf{T}_{[V]}$**-**$\mathbf{T}_{autor\_or}$ gerado pelo algoritmo é mostrado na Figura 6.9. A seguir, mostramos como foi gerado o código SQL:1999 para cada atributo de $\mathbf{T}_{[V]}$. No resto desta seção, seja $\mathbf{t}_{[V]}$ o objeto criado por esse construtor a partir do objeto base $\mathbf{t}_{autor\_or}$.

– O atributo nome (linha 1 - Figura 6.9) é um atributo monovalorado de valor atômico definido pelas AC $\psi_2^{[V]}$: [$\mathbf{T}_{[V]}$•**nome**] ≡ [$\mathbf{T}_{autor\_or}$•**nome**]. Do Caso 1 da Tabela 6-3, temos que $\mathbf{t}_{[V]}$•nome = $\mathbf{t}_{autor\_or}$•nome.
– O atributo foto (linha 2 - Figura 6.9) é um atributo monovalorado de valor atômico definido pela AC $\psi_3^{[V]}$: [$\mathbf{T}_{[V]}$•**foto**] ≡ [$\mathbf{T}_{autor\_or}$•**foto**]. Do Caso 1 da Tabela 6-3, temos que $\mathbf{t}_{[V]}$•foto= $\mathbf{t}_{autor\_or}$•foto.

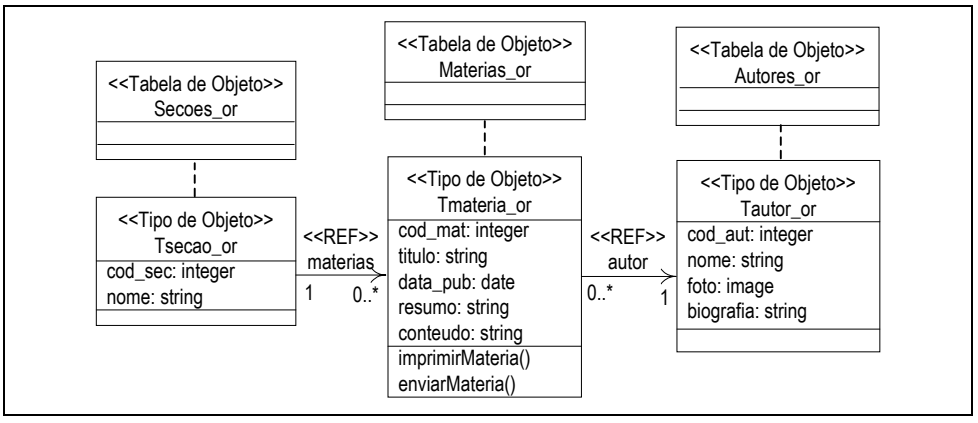


Figura 6.7: Banco de Dados objeto-relacional Publicações_or

VCN: $\mathcal{V}$		
Parâmetros: param: string		
Tabela Base: Autores_or	**Condição de Seleção:** cod_aut = param	**(ACE**: $\psi_1^{[V]}$: [V] ≡ [Autores_or[cod_aut = param]]**)**
Atributos: nome: string foto: string biografia: string secoes: set of <string> matérias: set of <$T_{[materia]}$> titulo: string data_publicacao: string resumo: string	**Assertivas de Correspondência:** $\psi_2^{[V]}$: [$T_{[V]}$•nome] ≡ [Tautor_or•nome] $\psi_3^{[V]}$: [$T_{[V]}$•foto] ≡ [Tautor_or•foto] $\psi_4^{[V]}$: [$T_{[V]}$•biografia] ≡ [Tautor_or•biografia] $\psi_5^{[V]}$: [$T_{[V]}$•secoes] ≡ [Tautor_or•autor^{-1}•materias^{-1}•nome] $\psi_6^{[V]}$: [$T_{[V]}$•materias] ≡ [Tautor_or•autor^{-1}] $\psi_7^{[V]}$: [$T_{[materia]}$•titulo] ≡ [Tmateria_or•titulo] $\psi_8^{[V]}$: [$T_{[materia]}$•data_publicacao] ≡ [Tmateria_or•data_pub] $\psi_9^{[V]}$: [$T_{[materia]}$•resumo] ≡ [Tmateria_or•resumo]	
Elos:		
Operações:		
Ordenação:		

Figura 6.8: Cartão de especificação da VCN $\mathcal{V}$, especificada sobre o banco de dados

- O atributo biografia (linha 3 - Figura 6.9) é um atributo monovalorado de valor atômico definido pela AC $\psi_4^{[V]}$: [$\mathbf{T}_{[V]}$•**biografia**] ≡ [$\mathbf{T}_{autor\_or}$•**biografia**]. Do Caso 1 da Tabela 6-3, temos que $\mathbf{t}_{[V]}$•biografia = $\mathbf{t}_{autor\_or}$•biografia.
- O atributo secoes (linhas 4 a 7 - Figura 6.9) é um atributo multivalorado de valor atômico definido pela AC $\psi_5^{[V]}$:[$\mathbf{T}_{[V]}$•**secoes**] ≡ [$\mathbf{T}_{autor\_or}$•**autor^{-1}•materias^{-1}•nome**], satisfazendo, assim, o Caso 6 da Tabela 6-3. Dado que φ = autor^{-1}•materias^{-1} é um caminho multivalorado, GeraConsulta(φ, $\mathbf{t}_{secao\_or}$) retorna: $\mathbf{S}_{\varphi}$="$\mathbf{t}_{secao\_or}$", $\mathbf{F}_{\varphi}$="Secoes_or $\mathbf{t}_{secao\_or}$" e $\mathbf{W}_{\varphi}$=" **REF**($\mathbf{t}_{autor\_or}$) **IN** (**SELECT** $\mathbf{t}_{materia\_or}$•autor **FROM TABLE**($\mathbf{t}_{secao\_or}$•materias) $\mathbf{t}_{materia\_or}$)". Assim, do Caso 8 da Tabela 6-3 temos que:

  $\mathbf{T}_{autor\_or}$•secoes=

  CURSOR (**SELECT** $\mathbf{t}_{secao\_or}$•nome
  FROM Secoes_or $\mathbf{t}_{secao\_or}$
  WHERE REF($\mathbf{t}_{autor\_or}$) **IN**
  (SELECT $\mathbf{t}_{materia\_or}$•autor **FROM TABLE(**$\mathbf{t}_{secao\_or}$•materias**)** $\mathbf{t}_{materia\_or}$**)) AS** secoes**)**

- O atributo materias (linhas 8 a 12 - Figura 6.9) é um atributo multivalorado de valor estruturado definido pela AC $\psi_6^{[V]}$:[$\mathbf{T}_{[V]}$•**materias**] ≡ [$\mathbf{T}_{autor\_or}$•**autor^{-1}**], onde $\mathbf{T}_{[materia]}$ é o tipo

do atributo materias e $\mathbf{T}_{materia\_or}$ é o tipo do caminho **autor**$^{-1}$. Do Caso 10 da Tabela 6-3, temos que:

$\mathbf{T}_{autor\_or}$•materias =

CURSOR (SELECT Construtor_$T_{[materia]}$-$T_{material\_or}$($\mathbf{S}_{\varphi}$)
FROM $\mathbf{F}_{\varphi}$
WHERE $\mathbf{W}_{\varphi}$),

onde GeraConsulta(φ, $\mathbf{t}_{materia\_or}$) retorna: $\mathbf{S}_{\varphi}$="$\mathbf{t}_{materia\_or}$", $\mathbf{F}_{\varphi}$="Materias_or $\mathbf{t}_{materia\_or}$" e $\mathbf{W}_{\varphi}$=" $\mathbf{t}_{materias\_or}$•autor **= REF**($\mathbf{t}_{autor\_or}$)) **AS** materias", dado que φ = autor^{-1} é um caminho multivalorado virtual.

Os valores dos atributos do construtor **Construtor_**$\mathbf{T}_{[materia]}$-$\mathbf{T}_{materia\_or}$ são definidos pelas ACC's de $\mathbf{T}_{[materia]}$&$\mathbf{T}_{materia\_or}$, como mostrado a seguir. No resto desta seção, considere $\mathbf{t}_{[materia]}$ o objeto criado pelo construtor a partir do objeto base em $\mathbf{t}_{materia\_or}$.

- O atributo titulo (linha 8 - Figura 6.9) é monovalorado de valor atômico definido pela ACC $\psi_7^{[V]}$:[$\mathbf{T}_{[materia]}$•**titulo**] ≡ [$\mathbf{T}_{materia\_or}$•**titulo**]. Do Caso 1 da Tabela 6-3, temos que: $\mathbf{t}_{[materia]}$•titulo = $\mathbf{t}_{materia\_or}$•titulo.
- O atributo data_pub (linhas 9 - Figura 6.9) é um atributo monovalorado de valor estruturado definido pela ACC $\psi_8^{[V]}$:[$\mathbf{T}_{[materia]}$•**data_publicacao**] ≡ [$\mathbf{T}_{materia\_or}$•**data_pub**]. Do Caso 1 da Tabela 6-3, temos que: $\mathbf{t}_{[materia]}$•data_pub = $\mathbf{t}_{materia\_or}$•data_pub.
- O atributo resumo (linhas 10 - Figura 6.9) é um atributo monovalorado de valor estruturado definido pela ACC $\psi_9^{[V]}$:[$\mathbf{T}_{[materia]}$•**resumo**] ≡ [$\mathbf{T}_{materia\_or}$• **resumo**]. Do Caso 1 da Tabela 6-3, temos que: $\mathbf{t}_{[materia]}$•resumo = $\mathbf{t}_{materia\_or}$•resumo.

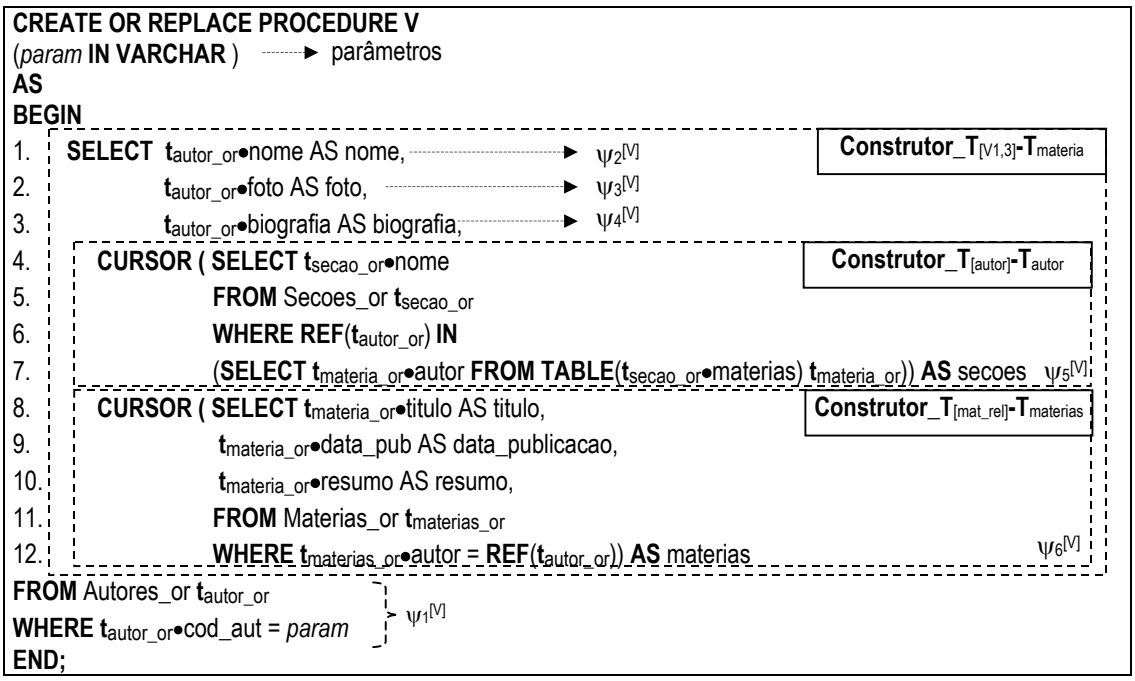


Figura 6.9: Definição da VCN $\mathcal{V}$ sobre o esquema do banco de dados Publicações_or

6.3 Algoritmo GeraAssertivas

O algoritmo GeraAssertivas gera, a partir das AC's da VCN com o banco e das AC´s da visão de objeto base com o banco, as AC´s da VCN com o Banco de dados. A seguir apresentamos as regras de inferência usada no algoritmo:

-- Regras de Inferência de ACE.

No resto desta seção, considere **V** uma VCN, **C** a sua visão de objeto base de **V**, **B** a tabela base de **C** e **p** um predicado condicional.

Regra ACE01:

Se [V] ≡ [C] e [C] ≡ [B] então [V] ≡ [B].

Regra ACE02:

Se [V] ≡ [C[p]] e [C] ≡ [B] então [V] ≡ [B[p']], onde p' corresponde ao predicado condicional p traduzido no banco de dados.

Regra ACE03:

Se $[V] \equiv [C]$ e $[C] \equiv [B[p]]$ então $[V] \equiv [B[p]]$.

Regra ACE04:

Se $[V] \equiv [C[p_1]]$ e $[C] \equiv [B[p_2]]$ então $[V] \equiv [B[p']]$, onde p' é a conjunção dos predicados p_1' e p_2, e p_1' é a tradução do predicado p_1 sobre o banco de dados.

-- Regras de Inferência de ACC.

Nas regras a seguir, sejam $\mathbf{T_v}$, $\mathbf{T_c}$ e $\mathbf{T_b}$ os tipos **V**, **C** e **B**, respectivamente.

Regra ACC01:

Se

1) $[T_v \bullet a] \equiv [T_c \bullet \varphi]$ é uma ACC de $\mathbf{T_v}$ & $\mathbf{T_c}$, onde **a** é um atributo de $\mathbf{T_v}$ e φ é um caminho de $\mathbf{T_c}$, tal que $\varphi = e_1 \bullet e_2 \bullet ... \bullet e_n$, como mostrado na Figura 6.10;

2) ψ_1 é uma ACC de $\mathbf{T_c}$ & $\mathbf{T_b}$ dada por ψ_1: $[T_c \bullet e_1] \equiv [T_b \bullet \varphi_1]$, onde $\mathbf{e_1}$ é um atributo de $\mathbf{T_c}$ e φ_1 é um caminho de $\mathbf{T_b}$; e

3) ψ_i: $[T_{e_{i-1}} \bullet e_i] \equiv [T_{\varphi_{i-1}} \bullet \varphi_i]$ é uma ACC de $\mathbf{T_{e_{i-1}}}$ & $\mathbf{T_{\varphi_{i-1}}}$, $1 < i \leq n$, onde $\mathbf{T_{e_{i-1}}}$ é o tipo do atributo $\mathbf{e_{i-1}}$, $\mathbf{e_i}$ é um atributo de $\mathbf{T_{e_{i-1}}}$, $\mathbf{T_{\varphi_{i-1}}}$ é o tipo do caminho φ_{i-1}, e φ_i é um caminho de $\mathbf{T_{e_{i-1}}}$.

então $[T_v \bullet a] \equiv [T_b \bullet \varphi_1 \bullet \varphi_2 \bullet ... \bullet \varphi_n]$

Regra ACC02:

Se

1) $[T_v \bullet a] \equiv [T_c \bullet \varphi]$ é uma ACC de $\mathbf{T_v}$ & $\mathbf{T_c}$, onde **a** é um atributo de $\mathbf{T_v}$ e φ é um caminho de $\mathbf{T_c}$, tal que $\varphi = e_1 \bullet e_2 \bullet ... \bullet e_n$, como definido na Figura 6.10;

2) ψ_1 é uma ACC de $\mathbf{T_c}$ & $\mathbf{T_b}$ dada por ψ_1: $[T_c \bullet e_1] \equiv [T_b \bullet \varphi_1]$, onde $\mathbf{e_1}$ é um atributo de $\mathbf{T_c}$ e φ_1 é um caminho de $\mathbf{T_b}$;

3) ψ_i: $[T_{e_{i-1}} \bullet e_i] \equiv [T_{\varphi_{i-1}} \bullet \varphi_i]$ é uma ACC entre $\mathbf{T_{e_{i-1}}}$ & $\mathbf{T_b}$, onde $1 < i < n$, $\mathbf{T_{e_{i-1}}}$ é o tipo do atributo $\mathbf{e_{i-1}}$, $\mathbf{e_i}$ é um atributo de $\mathbf{T_{e_{i-1}}}$, $\mathbf{T_{\varphi_{i-1}}}$ é o tipo do caminho φ_{i-1}, φ_i é um caminho

de $T_{e_{i-1}}$. Cada ψ_i especifica como o valor do atributo e_{i-1} pode ser sintetizado a partir do banco de dados; e

4) ψ_n: $[T_{e_{n-1}} \bullet e_n] \equiv [T_{\varphi_{n-1}} \bullet \varphi_n, \{a_{\varphi_{n1}}, a_{\varphi_{n2}}, ..., a_{\varphi_{nk}}\}]$ é uma ACC entre $T_{e_{n-1}}$ & T_b, onde $T_{e_{n-1}}$ é o tipo do atributo e_{n-1}, e_n é um atributo de $T_{e_{n-1}}$, $T_{\varphi_{n-1}}$ é o tipo do caminho φ_{n-1}, φ_n é um caminho de $T_{e_{n-1}}$, $a_{\varphi_{n1}}, a_{\varphi_{n2}}, ..., a_{\varphi_{nk}}$ são atributos de T_{φ_n}, onde $1 < i \leq k$. A ACC ψ_n especifica como o valor do atributo e_{n-1} pode ser sintetizado a partir do banco de dados.

então $[T_v \bullet a] \equiv [T_b \bullet \varphi_1 \bullet \varphi_2 \bullet ... \bullet \varphi_n, \{a_{\varphi_{n1}}, a_{\varphi_{n2}}, ..., a_{\varphi_{nk}}\}]$

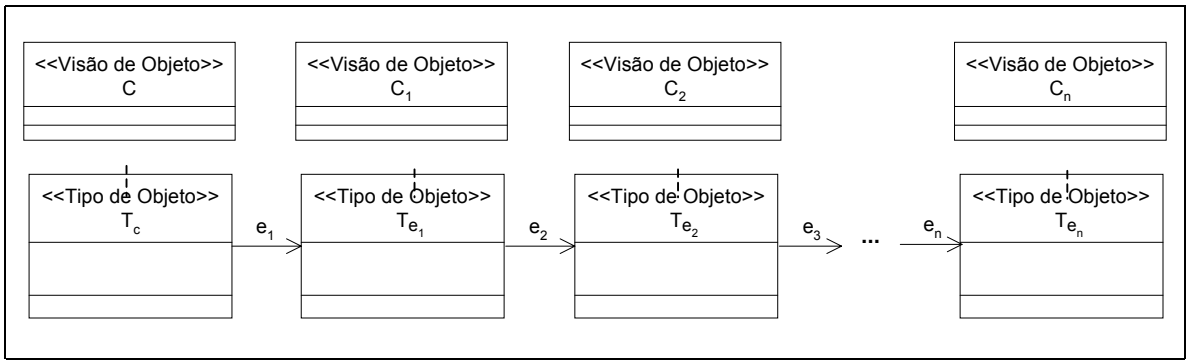


Figura 6.10: Caminho φ de T_c

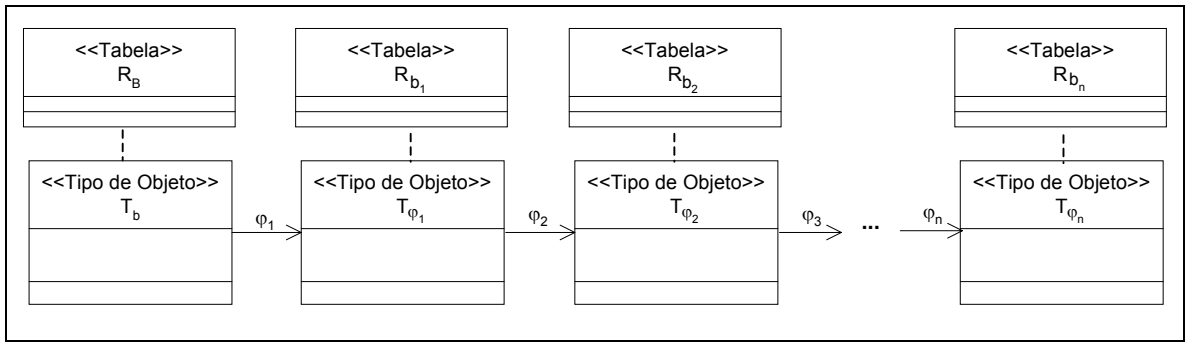


Figura 6.11: Caminho $\varphi_1 \bullet \varphi_2 \bullet ... \bullet \varphi_n$ de T_b

Considere, por exemplo, a VCN $\mathcal{V}$, cujo cartão de especificação é mostrado na Figura 6.12, definida sobre o esquema da visão Autores_v (Figura 6.13). A visão Autores_v é especificada sobre o esquema do banco de dados da Figura 5.1, e as AC's de T_{autor} & $T_{autores\_rel}$ são mostradas na Figura 6.14. As ACC's de T_V & $T_{autores\_rel}$ mostradas na Figura

6.15, que especificam como sintetizar o valor de cada atributo de $\mathbf{T}_V$ a partir do banco de dados, podem ser inferidas a partir das ACC's de $\mathbf{T}_V$ & $\mathbf{T}_{autor}$ (Figura 6.12) e de $\mathbf{T}_{autor}$ & $\mathbf{T}_{autores\_rel}$ (Figura 6.14).

VCN: $\mathcal{V}$	
Parâmetros: param: string	
Visão de Objeto Base: Autores_v	**Condição de Seleção:** cod_aut = param **(ACE**: $\psi_1^{[V]}$: [V] ≡ [Autores_v[cod_aut = param]]**)**
Atributos:	**Assertivas de Correspondência:**
nome: string	$\psi_2^{[V]}$: $[T_{[V]} \bullet nome] \equiv [T_{autor} \bullet nome]$
foto: string	$\psi_3^{[V]}$: $[T_{[V]} \bullet foto] \equiv [T_{autor} \bullet foto]$
biografia: string	$\psi_4^{[V]}$: $[T_{[V]} \bullet biografia] \equiv [T_{autor} \bullet biografia]$
secoes: set of <string>	$\psi_5^{[V]}$: $[T_{[V]} \bullet secoes] \equiv [T_{autor} \bullet secoes \bullet nome]$
matérias: set of <$T_{[materia]}$>	$\psi_6^{[V]}$: $[T_{[V]} \bullet matérias] \equiv [T_{autor} \bullet materias]$
titulo: string	$\psi_7^{[V]}$: $[T_{[materia]} \bullet titulo] \equiv [T_{materia} \bullet titulo]$
data_pub: string	$\psi_8^{[V]}$: $[T_{[materia]} \bullet data\_pub] \equiv [T_{materia} \bullet data\_pub]$
resumo: string	$\psi_9^{[V]}$: $[T_{[materia]} \bullet resumo] \equiv [T_{materia} \bullet resumo]$
Elos:	
Operações:	
Ordenação:	

Figura 6.12: Cartão de Especificação da VCN $\mathcal{V}$

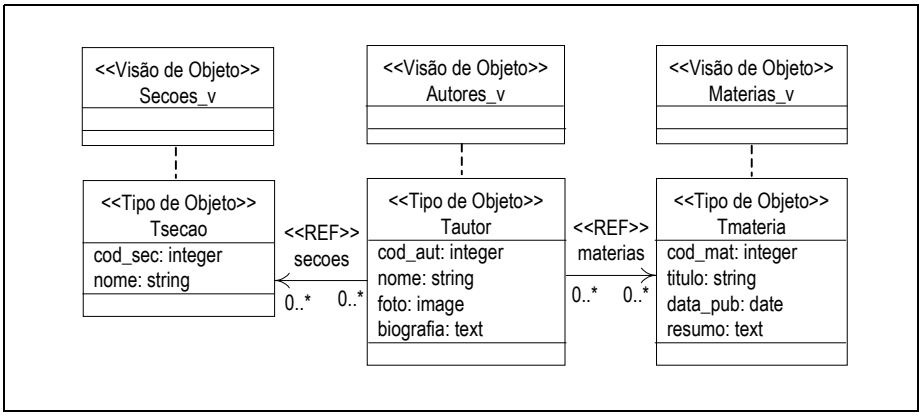


Figura 6.13: Esquema da visão Autores_v

ACE de **Autores_v** & **Autores_rel**
ψ_1: [Autores_v] ≡ [Autores_rel]

ACC's de $\mathbf{T}_{[V]}$ & $\mathbf{T}_{autores\_rel}$
ψ_2: [T_{autor}•cod_aut] ≡ [$T_{autores\_rel}$ •cod_aut]
ψ_3: [T_{autor}•nome] ≡ [$T_{autores\_rel}$ •nome]
ψ_4: [T_{autor}•foto] ≡ [$T_{autores\_rel}$ •foto]
ψ_5: [T_{autor}•biografia] ≡ [$T_{autores\_rel}$ •biografia]
ψ_6: [T_{autor}•secao] ≡ [$T_{autores\_rel}$•fk_1^{-1}•fk_4]
ψ_7: [T_{autor}•materias] ≡ [$T_{autores\_rel}$•fk_1^{-1}]

ACC's de $\mathbf{T}_{secao}$ & $\mathbf{T}_{secoes\_rel}$
ψ_8: [$\mathbf{T}_{secao}$•cod_sec] ≡ [$\mathbf{T}_{secoes\_rel}$•titulo]
ψ_9: [$\mathbf{T}_{secao}$•nome] ≡ [$\mathbf{T}_{secoes\_rel}$•nome]

ACC's de $\mathbf{T}_{materia}$ & $\mathbf{T}_{materias\_rel}$
ψ_{10}: [$\mathbf{T}_{materia}$•titulo] ≡ [$\mathbf{T}_{materias\_rel}$•titulo]
ψ_{11}: [$\mathbf{T}_{materia}$•data_pub] ≡ [$\mathbf{T}_{materias\_rel}$•data_pub]
ψ_{12}: [$\mathbf{T}_{materia}$•resumo] ≡ [$\mathbf{T}_{materias\_rel}$•resumo]

Figura 6.14: AC's de Autores_v com o banco de dados

ACE de **V** & **Autores_rel**
δ_1: [**V**] ≡ [**Autores_rel**[cod_aut = param]] (de $\psi_1^{[V]}$ e ψ_1)

ACC's de $\mathbf{T}_{[V]}$ & $\mathbf{T}_{autores\_rel}$
δ_2: [$\mathbf{T}_{[V]}$•nome] ≡ [$\mathbf{T}_{autores\_rel}$•nome] (de $\psi_2^{[V]}$ e ψ_3)
δ_3: [$\mathbf{T}_{[V]}$•foto] ≡ [$\mathbf{T}_{autores\_rel}$•foto] (de $\psi_3^{[V]}$ e ψ_4)
δ_4: [$\mathbf{T}_{[V]}$•biografia] ≡ [$\mathbf{T}_{autores\_rel}$•biografia] (de $\psi_4^{[V]}$ e ψ_5)
δ_5: [$\mathbf{T}_{[V]}$•secoes] ≡ [$\mathbf{T}_{autores\_rel}$•fk_1^{-1}•fk_4•nome] (de $\psi_5^{[V]}$, ψ_6 e ψ_9)
δ_6: [$\mathbf{T}_{[V]}$•materias] ≡ [$\mathbf{T}_{autores\_rel}$•fk_1^{-1}] (de $\psi_6^{[V]}$ e ψ_7)

ACC's de $\mathbf{T}_{[materia]}$ & $\mathbf{T}_{materia\_rel}$
δ_7: [$\mathbf{T}_{[materia]}$•titulo] ≡ [$\mathbf{T}_{materias\_rel}$•titulo] (de $\psi_7^{[V]}$ e ψ_{10})
δ_8: [$\mathbf{T}_{[materia]}$•data_pub] ≡ [$\mathbf{T}_{materias\_rel}$•data_pub] (de $\psi_8^{[V]}$ e ψ_{11})
δ_9: [$\mathbf{T}_{[materia]}$•resumo] ≡ [$\mathbf{T}_{materias\_rel}$•resumo] (de $\psi_9^{[V]}$ e ψ_{12})

Figura 6.15: AC's de $\mathbf{T}_{[V]}$ & $\mathbf{T}_{autores\_rel}$

6.4 Algoritmo GeraEstilo

Nesta seção, apresentamos o algoritmo GeraEstilo, que gera o estilo XSL contendo as regras para transformar o documento XML canônico, resultante do processamento de uma página XSQL, no documento XML com formato do esquema XML da VCN. A partir do esquema XML de uma VCN (X_V), o algoritmo gera um documento XSLT, que reestrutura o documento XML canônico, de acordo com a estrutura de X_V. A Figura 6.16 mostra o padrão para esquemas XML das VCN's. Esse esquema possui um nó raiz, que apresenta um único elemento primário, de acordo com a estrutura das VCN's. O mapeamento automático do documento XML Canônico em um documento com a estrutura da VCN é possível porque é conhecida a estrutura do documento XML Canônico resultante do processamento das páginas XSQL.

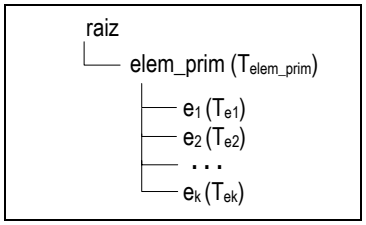


Figura 6.16: Padrão de esquema XML das VCN

Dado o esquema XML X_V de uma VCN no formato apresentado na Figura 6.16, o algoritmo **GeraEstilo** gera um documento XSLT, com a estrutura descrita na Figura 6.17. O conteúdo do elemento primário **elem_prim**, ou seja, #e_1# #e_2# ... #e_k# é gerado pelo algoritmo **GeraRegras**. Esse algoritmo recebe como entrada a declaração do tipo complexo $\mathbf{T}_{elem\_prim}$ do elemento elem_prim e, para cada declaração de elemento **e** em $\mathbf{T}_{elem\_prim}$, gera as regras para obter o conteúdo do elemento e, de acordo com os casos descritos na Tabela 6-4.

```
<?xml version="1.0" encoding="UTF-8"?>">
<xsl:stylesheet version  = "1.0" xmlns:xsl = "http://www.w3c.org/1999/XSL/Transform"
<xsl:template match = ".">
      <xsl:element name = "Resultado">
          <xsl:for-each select = "ROWSET/ROW">
                    <xsl:element name = "elem_prim">
                              # e1#
                              # e2#
                              ...
                              # ek#
                    <xsl:element>
          </xsl:for-each>
      <xsl:element>
</xsl:template>
```

Figura 6.17: Documento XSLT gerado pelo algoritmo GeraEstilo

Casos	#e#
1. **e** é monoocorrência e T_e é simples	<xsl:element name= "**e**"> <xsl:value-of select="**e**"/> </xsl:element>
2. **e** é monoocorrência e T_e é complexo	<xsl:for-each select= "**e/***"> <xsl:element name= "**e**"> GeraRegras ($\mathbf{T}_e$) </xsl:element> </xsl:for-each>
3. **e** é multiocorrência e T_e é simples	<xsl:for-each select= "**e/***"> <xsl:element name= "**e**"> <xsl:value-of select= "."/> </xsl:element> </xsl:for-each>
4. **e** é multiocorrência e T_e é complexo	<xsl:for-each select= "**e**"> <xsl:element name= "**e**"> GeraRegras ($\mathbf{T}_e$) </xsl:element> </xsl:for-each>

Tabela 6-4: Casos do algoritmo GeraRegras

Exemplo 6.5

Considere a VCN $\mathcal{V}_{1,3}$ do UID $\mathcal{V}_{1,3}$ (Figura 4.2), cujo esquema XML é apresentado na Figura 6.18. O estilo XSLT gerado para $\mathcal{V}_{1,3}$ pelo algoritmo GeraEstilo é mostrado na Figura 6.19. Dado o esquema XML de $\mathcal{V}_{1,3}$, o algoritmo GeraEstilo gera as linhas 1 a 5 e 32 a 35 do estilo de $\mathcal{V}_{1,3}$, que correspondem ao elemento primário Materia desse esquema XML. Esse algoritmo faz uma chamada ao procedimento GeraRegras para gerar o conteúdo de cada elemento de $T_{[V_{1,3}]}$. A seguir, discutimos como é gerado o conteúdo desses elementos.

- O elemento titulo é um elemento monocorrência de tipo simples (string). Do Caso 1 da Tabela 6-4, obtemos as linhas 6 a 8.

- O elemento data_publicacao é um elemento monocorrência de tipo simples (string). Do Caso 1 da Tabela 6-4, obtemos as linhas 9 a 11.

- O elemento conteudo é um elemento monocorrência de tipo simples (string). Do Caso 1 da Tabela 6-4, obtemos as linhas 12 a 14.

- O elemento autor é um elemento monocorrência de tipo complexo ($T_{[autor]}$). Do Caso 2 da Tabela 6-4, obtemos as linhas 15 a 21.Observe que para cada elemento definido no tipo $T_{[autor]}$, é feita uma chamada recursiva ao procedimento GeraRegras.

- O elemento mat_rel é um elemento multiocorrência de tipo complexo ($T_{[mat\_rel]}$). Do Caso 4 da Tabela 6-4, obtemos as linhas 22 a 31.Observe que para cada elemento definido no tipo $T_{[mat\_rel]}$, é feita uma chamada recursiva ao procedimento GeraRegras.

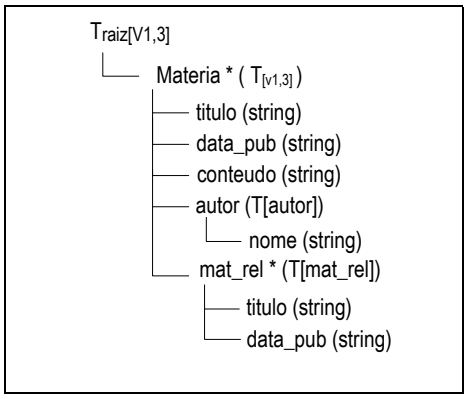


Figura 6.18: Esquema XML de $\mathcal{V}_{1,3}$

```
1. <xsl:stylesheet version = “1.0” xmlns:xsl = “http://www.w3c.org/1999/XSL/Transform">
2. <xsl:template match = “.”>
3.      <xsl:element name = “Resultado”>
4.           <xsl:for-each select = “ROWSET/ROW”>
5.                <xsl:element name = “Materia”>
6.                     <xsl:element name = “titulo”>
7.                          <xsl:value-of select = “TITULO”/>
8.                     </xsl:element>
9.                     <xsl:element name = “data_publicacao”>
10.                         <xsl:value-of select = “DATA_PUB”/>
11.                    </xsl:element>
12.                    <xsl:element name = “conteudo”>
13.                         <xsl:value-of select = “CONTEUDO”/>
14.                    </xsl:element>
15.                    <xsl:for-each select = “AUTOR/*”>
16.                         <xsl:element name = “autor”>
17.                              <xsl:element name = nome>
18.                                   <xsl:value-of select = “NOME”/>
19.                              <xsl:element>
20.                         </xsl:element>
21.                    <xsl:for-each>
22.                    <xsl:for-each select = “MAT_REL/*”>
23.                         <xsl:element name = “mat_rel”>
24.                              <xsl:element name = titulo>
25.                                   <xsl:value-of select = “TITULO”/></titulo>
26.                              <xsl:element>
27.                              <xsl:element name = data_publicacao>
28.                                   <xsl:value-of select = “DATA_PUB”/></titulo>
29.                              </xsl:element>
30.                         </xsl:element>
31.                    </xsl:for-each>
32.               </xsl:element>
33.          </xsl:for-each>
34.     </xsl:element>
35. </xsl:template>
```

Figura 6.19: Estilo gerado para transformar esquema XML Canônico de $\mathcal{V}_{1,3}$

Capítulo 7
Conclusões e Trabalhos Futuros

Neste trabalho, propomos um processo para geração e manutenção das Visões de Contexto de Navegação (VCN's) para aplicações DIWA. As VCN's definem os requisitos de conteúdo dinâmico de cada página de uma aplicação *web*, e são especificadas conceitualmente sobre as visões de objeto, através de um conjunto de assertivas que especificam formalmente a correspondência do esquema da VCN com o esquema da visão de objeto base.

As visões de objeto da aplicação também são especificadas conceitualmente, através de um conjunto de assertivas que especificam formalmente as correspondências entre o esquema da visão de objeto e o esquema do banco de dados. A vantagem de especificar as VCN´s sobre as visões de objeto é que a especificação gerada é independente do esquema do banco de dados, e assim mudanças do esquema do banco não afetam as especificações conceituais das VCN´s.

Como mostrado no Capítulo 5, o uso das assertivas de correspondência para especificação das VCN's e das visões de objeto permite que a implementação das VCN's seja realizada de modo automático. Neste trabalho, discute-se a implementação das VCN's como visões de objeto e como visões XML. Em ambos os enfoques, as VCN's podem ser definidas sobre o esquema do banco de dados ou sobre uma visão de objeto. Este último enfoque tem a vantagem de uma maior independência das VCN's em relação a modificações no esquema do banco de dados.

Conforme foi observado, o uso de páginas dinâmicas para publicar o resultado de consultas SQL no formato XML é bastante eficiente. Entretanto, gerar essas páginas manualmente não é tarefa simples para aplicações DIWA, que geralmente são constituídas de grande número de VCN's. No enfoque proposto, as páginas XSQL que implementam as

VCN's podem ser geradas de forma automática, a partir das especificações conceituais das VCN's.

As atividades do ProDIWA podem ser incorporadas aos métodos de desenvolvimento de *software* para *Web* já existentes, como o OOHDM e o OO-H, 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 VCN's. Sem um método para especificação formal das VCN's, não é possível automatizar a sua implementação e manutenção.

Outra contribuição deste trabalho é o ambiente proposto para apoiar as diversas atividades do processo ProDIWA. Como foi apresentado no Capítulo 6, esse ambiente é composto de várias ferramentas, as quais são classificadas em: Ferramentas de Projeto, que apóiam as atividades Projeto das Visões de Objetos e Projeto das VCN's; e Ferramentas de Implementação, que apóiam as atividades Implementação e Manutenção das VCN's. Além da ferramenta VBA, proposta em [42], as seguintes ferramentas já foram implementadas:

(i) Ferramenta para Especificação das Visões de Objeto, que provê uma interface gráfica para especificação das visões de objeto;

(ii) Ferramenta para Especificação das VCN's, que provê uma interface gráfica para especificação das VCN's.

Como trabalhos futuros, pretendemos:

- Dar continuidade à implementação do ambiente ProDIWA. As seguintes ferramentas faltam ser implementadas:

  (i) Ferramenta para Geração da Definição SQL das VCN´s, onde as consultas SQL são geradas a partir das AC's da VCN e podem ser definidas sobre o esquema de uma visão de objeto ou de um banco de dados (relacional ou objeto-relacional);

  (ii) Ferramenta para Geração de Página Dinâmica;

  (iii) Ferramenta para Geração de Estilo XSLT, o qual gera o estilo XSLT a partir do esquema XML da VCN;

(iv) Ferramenta para Geração da Definição de Visão XML, onde a consulta XQuery que define a visão XML da aplicação é gerada a partir das assertivas da visão;

(v) Ferramenta para Geração da Definição *XQuery* das VCN´s;

(vi) Ferramenta para Geração do Esquema da Visão XML da aplicação**,** a partir do esquema das visões de objeto da aplicação e suas assertivas de correspondências com o banco de dados;

- Estender a Ferramenta para Geração da Definição SQL das VCN´s para:
    - suportar outros tipos de assertivas de correspondência, de forma a permitir a geração de novos tipos de consultas SQL;
    - permitir criação de consultas de atualização no banco de dados, para tratar VCN's de atualização;
- Propor um método para projeto de banco de dados relacional e objeto-relacional a partir da integração das visões de objetos; de forma que as assertivas de correspondências das visões de objeto com o banco de dados sejam geradas pelas regras de mapeamento propostas pelo método.
- Estender o ProDIWA para tratar aplicações web cujos conteúdos das VCN's são obtidos da integração de múltiplas bases de dados, as quais podem ser heterogêneas e distribuídas.

Referências Bibliográficas

[1] AMBLER, S. W.. *Mapping Objects to Relational Databases*. An AmbySoft Inc. White Paper. 26 de fevereiro de 1999. Disponível em: <http://www.AmbySoft.com/mappingObjects.pdf>.

[2] ATZENI, P.; MECCA, G.; MERIALDO, P.. *Design and Maintenance of Data-Intensive Web Sites*. Proceedings of International Conference on Extending Database Technology, EDBT98, Espanha, 1998.

[3] ATZENI, P.; MECCA, G.; MERIALDO, P.; CRESCENZI, V.. *The Araneus Guide to Web-Site Development*. Araneus Project Work Report, Versão 1.0, Março, 1999.

[4] BERNERS-LEE, T.; HENDLER, J.; LASSILA, O.. *The Semantic Web*. Scientific American, Maio, 2001.

[5] BOAS, R. M. F. V.. *XMLS+ Matcher: Um Método para Matching de XML Schemas Semânticos*. /Dissertação de Mestrado - Universidade Federal do Ceará, 2002/.

[6] CAREY, M.; et al. *XPERANTO*: Publishing object-relational data as XML. In: *Proceedings Third International Workshop on the Web and Databases*. Dallas, Texas: Maio 2000, págs. 105-110.

[7] CARLSON, D.. *Modeling XML Applications with UML*. Addison-Wesley, 2001.

[8] CERI, S.; FRATERNALI, P.; BONGIO, A.; BRAMBILLA, M.; COMAI, S.; MATERA, M.. *Designing Data-Intensive Web Applications*. Morgan Kaufmann Publishers, 2003.

[9] CHAMBERLIN, D.; ROBIE, J.; FLORESCU, D.. Quilt: An XML Query Language for Heterogeneous Data Source. In: *Proceedings of the Workshop on Web and*

Databases (WebDb). In Conj. with SIGMOD'00. Dallas, Texas: Addison-Wesley, Maio, 2000.

[10] CHAUDHRI, A. B.; ZICARI, R.. *Succeeding with Object Databases: A Practical Look at Today's Implementations with Java and XML*. John Wiley & Sons, Setembro, 2000.

[11] *Code Charge Studio*. Disponível em <http://www.codecharge.com>. Acessado em 30 de agosto de 2004.

[12] CONALLEN, J. *Building Web Applications with UML*. Second Edition. Addison-Wesley, 2003.

[13] COSTA, J. P.. *Atualização de Bancos de Dados Objeto Relacionais Através de Visões de Objetos*. Fortaleza. /Dissertação de Mestrado em Ciência da Computação - Universidade Federal do Ceará – UFC/. Dezembro de 2002.

[14] DEUTSCH, A.; Fernandez, M.; FLORESCU, D.; LEVY, A.; SUCIU, D.. *XML-QL: A Query Language for XML*. 8 th International WWW Conference, Toronto, Maio de 1999.

[15] DUCKETT, J.; GRIFFIN, O.; MOHR, S.; NORTON, F.; STOKES-REES, I.; WILLIAMS, K.; CAGLE, K.; OZU, N.; TENNISON, J.. *Professional XML Schemas*. Wrox Press Ltd. Birmingham, 2001.

[16] ELMASRI, R.; NAVATHE, S. B. *Fundamentals of Database Systems*. Third Edition. Addison-Wesley, 2000.

[17] *Enterprise Architect – XML Schema Generation*. Versão 4.0. Copyright © Sparx Systems. Disponível em: <http://www.sparxsystems.com.au/xml_schema_generation.htm>. Acessado em 11 de abril de 2004.

[18] FAHL, G.; RISCH, T.. Query processing over object views of relational data. In: *The VLDB Journal*. 1997, págs. 261-281.

[19] FERNANDEZ, M.; KADIYSKA, Y.; MORISHIMA, A.; TAN, W.; SUCIU, D.. Silkroute: A Framework for Publishing Relational Data. In: *XML. ACM Transactions on Database System*. v. 27, n. 4, Dezembro, 2002, págs. 438-493.

[20] FERNÁNDEZ, M.; TAN, W.; SUCIU, D.. Silkroute: Trading between relations and XML. In: *Proceedings of the Ninth International World Wide Web Conference, (WWW'9)*. Amsterdam: Maio, 2000.

[21] FLORESCU, D.; DEUTSCH, A.; LEVY, A.; SUCIU, D.; FERNANDEZ, M.. A Query Language for XML. In: *Proceedings of Eighth International Conference on World Wide Web*. Toronto, Canadá, 1999, págs. 1155-1169.

[22] FRATERNALI, P.; PAOLINI, P.. *Model-Driven Development of Web Applications: The Autoweb System*. ACM Transaction on Information Systems, v. 28, n. 4, Out. 2001.

[23] GARZOTTO, F.; SCHWABE, D.; PAOLINI, P.. *HDM – A Model Based Approach to Hypermedia Application Design*. ACM Transaction on Information Systems, v. 11, Janeiro, 1993.

[24] GÓMEZ, J.; CACHERO, C.; PASTOR, O.. *Conceptual Modeling of Device-Independent Web Applications*. IEEE Multimedia, v. 8, n. 2, 2001.

[25] GÜELL, N.; SCHWABE, D.; VILAIN, P.. *Modeling Interactions and Navigations in Web Applications*. Lecture Notes in Computer Science 1921, Proceedings of the World Wild Web and Conceptual Modeling'00 Workshop, ER'00 Conference, Springer, Salt Lake City, 2000.

[26] *IBM XML for Tables*. Disponível em: <http://www.alphaworks.ibm.com/tech/xtable>. Acessado em 03 de março de 2004.

[27] LEE, D.; CHU, W.. Comparative Analysis of Six XML Schema Language. In: *ACM SIGMOD Record*. vol. 29. Setembro, 2000.

[28] LIMA, F.; SCHWABE, D.. *Application Modeling for the Semantic Web*. First Latin American Web Congress (LA-WEB'03) November 10 - 12, 2003 Santiago, Chile. p. 93.

[29] MADHAVAN, J.; BERNESTEIN, P.; RAHM, E.. Generic Schema Matching with Cupid. In: *Proc. Of VLDB*. p. 49-58, 2001.

[30] MARINHO, F. G.. *Padrões para a Modelagem Conceitual com a UML*. /Dissertação de Mestrado - Universidade Federal do Ceará/ 2001.

[31] *Microsoft Corporation*. Disponível em: <http://www.microsoft.com>. Acessado em 01 de Setembro de 2004.

[32] MUENCH, S.. *Building Oracle XML Applications*. O'Reilly, Setembro, 2000.

[33] O. Corporation. *Oracle9i - Application Developer's Guide - Object-Relational Features*. vol. Release 2 (9.2) - A96594-01, Março de 2002. Disponível em: <http://download-west.oracle.com/docs/cd/B10501_01/appdev.920/a96594.pdf>. Acessado em 7 de maio de 2004.

[34] O. Corporation. *Oracle XML Developer's Kit*. Disponível em: <http://otn.oracle.com/tech/xml/xdkhome .html> Acessado em 07 de Maio de 2004.

[35] *Oracle Corporation*. Disponível em: <http://technet.oracle.com>. Acessado em 7 de Maio de 2004.

[36] *Oracle Developer Suite*. Copyrigth © 2003, 2004 Oracle. Disponível em: <http://download.oracle.com/docs/pdf/B10668_05.pdf>. Acessado em 1 de Setembro de 2004.

[37] *PHP* . Disponível em: < http://www.php.org>. Acessado em 01 de Setembro de 2004.

[38] RAHM, E.; BERNSTEIN, P.. *On Matching Schemas Automatically*. Technical Report MST-TR-2001-17, 2001.

[39] *Rational Rose*. Versão 2003. Copyright © IBM. Disponível em: <http://www-306.ibm.com/software/rational>. Acessado em 30 de Agosto de 2004.

[40] ROSSI, G. H. *Um Método Orientado a Objetos para o Projeto de Aplicações Hipermídia*. /Tese de Doutorado - DI/PUC-RIO - Rio de Janeiro/ 1996.

[41] SAHUGUET, A.. Everything you ever wanted to know about dtds, but were afraid to ask. International Workshop on the Web and Databases. In: *Proceedings Of WebDB'2000*, pags. 69-74.

[42] SANTOS, L. A. de L.. *XML Publisher: Um Framework para Publicar Dados Objeto Relacionais em XML*. /Dissertação de Mestrado - Universidade Federal do Ceará/ 2004.

[43] SCHWABE, D; MEDEIROS, A. P. de. *Engenharia de Aplicações Web*. Notas de Aula, 2002.

[44] SCHWABE, D.; ROSSI, G.. *The Object-Oriented Hypermedia Design Model, Communications of the ACM*. Agosto, 1995.

[45] SCHWABE, D.; ROSSI, G.. *An Object Oriented Approach to Web-Based Application Design*. Theory and Practice of Object Systems 4(4), 1998. Wiley and Sons, New York, ISSN 1074-3224.

[46] SHANMUGASUNDARAM, J.; et al. *XPERANTO: Bridging Relational Technology and XML*. IBM Research Report, Junho, 2001.

[47] SHANMUGASUNDARAM, J.; et al. Querying XML Views of Relational Data. In: *Proceeding of 27th VLDB Conference*. Roma, Itália: 2001, págs. 261-270.

[48] Sun Microsystems. *Java Servlets Technology*. Disponível em: < http://java.sun.com/products/servlet/>. Acessado em 15 de Setembro de 2004.

[49] Sun Microsystems. *JavaServer Pages Technology*. Disponível em: < http://java.sun.com/products/jsp/>. Acessado em 15 de Setembro de 2004.

[50] VILAIN, P.; SCHWABE, D.; SIECKENIUS, C. A.. *Diagrammatic Tool for Representing User Interaction in UML*. Lecture Notes in Computer Science, Proc. UML'2000, York, 2000.

[51] TIDWELL, D.. *XSLT – Mastering XML Transformations*. First Edition. O'Reilly, Agosto, 2001.

[52] *WebRatio Site Development Studio*. Disponível em: <http://www.webratio.com>.

[53] WIDERHOLD, G.; BARSALOU, T.; LEE, B. S.; SIAMBELA, N.; SUJANSKY, W.. *Use of relational storage and a semantic model to generate objects: The Peguin Project*. Database'91: Merging Policy, Standards and Technology, 1991, pp. 503-515.

[54] WIEDERHOLD, G. Views, Objects, and Databases. In: *IEEE Computer*. Vol.19 No.12, December 1986, págs. 37-44.

[55] *World Wide Web Consortium*. Disponível em: <http://www.w3.org>. Acessado em 7 de janeiro de 2004.

[56] *World Wide Web Consortium*. *Extensible Markup Language (XML) Version 1.1*. W3C Recommendation, 4th February 2004. Disponível em: <http://www.w3.org/TR/2004/REC-xml11-20040204/>. Acessado em 10 de maio de 2004.

[57] *World Wide Web Consortium. HTML Version 4.0.1*. W3C Recommendation, 24 de dezembro de 1999. Disponível em: <http://www.w3.org/TR/html4/>. Acessado em 4 de dezembro de 2003.

[58] *World Wide Web Consortium*. *MathML 2.0*. W3C Recommendation, 21 de fevereiro de 2001. Disponível em: <http://www.w3.org/TR/MathML2/>. Acessado em 14 de março de 2004.

[59] *World Wide Web Consortium*. *Resource Description Framework* (RDF): Concepts and Abstract Syntax. W3C Recommendation, 10 de fevereiro de 2004. Disponível em: <http://www.w3.org/TR/rdf-concepts/>. Acessado em 20 de março de 2004.

[60] *World Wide Web Consortium*. *XML Path Language (XPath) Version 1.0*. W3C Recommendation, 16 de novembro de 1999. Disponível em: <http://www.w3.org/TR/xpath>. Acessado em 21 de abril de 2004.

[61] *World Wide Web Consortium*. *XML Schema Part 1: Structures*. W3C Recomendation, 02 de maio de 2001. Disponível em: <http://www.w3.org/TR/xmlschema-1/>. Acessado em 15 de outubro de 2003.

[62] *World Wide Web Consortium*. *XML Schema Part 2: Datatypes*. W3C Recomendation, 02 de maio de 2001. Disponível em: <http://www.w3.org/TR/xmlschema-2/>. Acessado em 17 de agosto de 2003.

[63] *World Wide Web Consortium*. *XQuery 1.0: An XML Query Language*. W3C Working Draft, 23 de julho de 2004. Disponível em: <http://www.w3.org/TR/xquery/>. Acessado em 8 de abril de 2004.

[64] *World Wide Web Consortium*. *Extensible StyleSheet Language Tansformation (XSLT)*. Version 1.0. W3C Recommendation, 16 de novembro de 1999. Disponível em: <http://www.w3c.org/TR/xslt>. Acessado em 7 de maio de 2004.

[65] ZEDULKA, J.. Object-relational modeling in UML. In: *Proceedings of the Conference*. Information Systems Modelling, Ostrava, CZ, MARCH, 2001, p. 17-24.

Apêndice A
Estudo de Caso

Este estudo de caso foi construído para demonstrar o uso do processo ProDIWA na geração das VCN's para uma aplicação denominada "Publicações On-*Line*", na qual os usuários podem acessar diversas informações sobre as matérias publicadas. A seguir, descrevemos os passos do processo para especificação e implementação das VCN's.

A1. Levantamento de Requisitos

Na atividade Levantamento de Requisitos, são identificados os atores e as tarefas a serem realizadas pela aplicação. As tarefas são descritas através de cenários e casos de uso. UID's são utilizados para representar graficamente as interações dos usuários com o sistema para execução de tarefas descritas em casos de uso. Os cenários, casos de uso e UID's utilizados neste estudo de caso foram extraídos de [43]. A atividade Levantamento de Requisitos consiste nos seguintes passos:

Passo 1: Identificação de atores e tarefas

- **Autor:** Leitor
- **Tarefas:**
  - Ler matérias de uma seção
  - Consultar informações de um autor
  - Consultar matérias favoritas

Passo 2: Especificação dos cenários

- Ator Leitor: Usuário 1

  Cenário 1.1: Consultar informações de um autor
  Descrição:
  Gostaria de obter informações sobre os autores das matérias. Para isto, eu informo o nome do autor e a aplicação me retorna o nome completo, a foto, a biografia do autor, as seções onde ele publica e uma lista das matérias publicadas por ele. Esta lista de matérias deve apresentar o título, a data de publicação e o resumo da matéria.

  Cenário 1.2: Ler matérias sobre esportes
  Descrição:
  Quero encontrar matérias sobre tênis. Para isso, eu seleciono a seção esportes e espero obter uma lista com as matérias existentes sobre este assunto. Caso exista uma matéria relacionada ao tênis, eu a seleciono e a aplicação retorna o texto da matéria e o nome do autor. Seria interessante ter as opções de imprimir a matéria e de enviá-la a um amigo.

- Ator Leitor: Usuário 2

  Cenário 2.1: Ler matérias de uma seção
  Descrição: Escolho uma seção e a aplicação me retorna todas as matérias dessa seção. A lista de matérias deve conter a data de publicação, o título e um resumo de cada matéria. Se tiver interesse em ler a matéria completa, eu a seleciono e vejo o seu conteúdo e o nome de seu autor. Quero ver também, caso exista, uma lista de matérias relacionadas à matéria que eu selecionei, e ter a possibilidade de acessar informações sobre o autor, como seu nome completo, foto e biografia.

  Cenário 2.2: Consultar minhas matérias favoritas
  Descrição: Informo meu e-mail e senha e a aplicação apresenta uma relação das matérias que eu selecionei anteriormente. Nessa relação, aparece o título, resumo e a data em que a matéria foi publicada. Seleciono uma das matérias apresentadas e vejo o seu conteúdo, nome do seu autor e os comentários que acrescentei sobre ela.

Passo 3: Especificação dos casos de uso

Deve-se agrupar os cenários que estão relacionados com uma mesma tarefa em um caso de uso. Para cada tarefa estabelecida, é criado um caso de uso.

- **Caso de Uso 1:** Ler matérias de uma seção
  Cenários: 1.2, 2.1
  Descrição:
  1. O usuário seleciona uma das seções apresentadas pela aplicação
  2. A aplicação retorna uma lista das matérias da seção selecionada, contendo a data de publicação, o título e o resumo de cada matéria
  3. O usuário seleciona a matéria de interesse

4. A aplicação retorna o título, a data de publicação, o conteúdo da matéria, o nome do autor e uma lista das matérias relacionadas, caso exista. Se desejar, o usuário pode imprimir a matéria ou enviá-la para um amigo
5. O usuário solicita informações sobre o autor
6. A aplicação retorna o nome, a foto e a biografia do autor

Alternativa:
No item 5, o usuário pode solicitar ver as informações detalhadas de uma matéria relacionada. Nesse caso, retorna para o item 4.

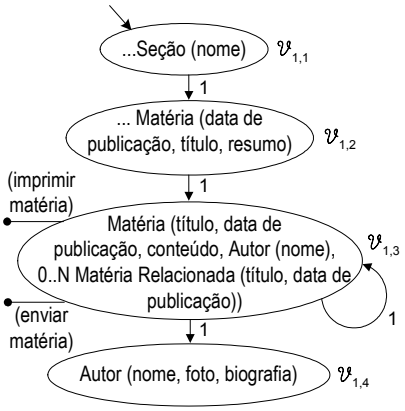
- **Caso de Uso 2:** Consultar matérias favoritas
  Cenário: 2.2
  Descrição:
  1. O usuário informa e-mail e senha
  2. Se a senha for válida, a aplicação retorna uma lista das matérias que possuem a palavra chave em uma das opções selecionadas, contendo o título, a data de publicação e o resumo de cada matéria. Senão, retorna mensagem de *login* inválido
  3. O usuário seleciona a matéria de interesse
  4. A aplicação retorna o título, a data de publicação, o conteúdo da matéria, o nome do autor e os comentários existentes sobre a matéria selecionada

- **Caso de Uso 3:** Consultar informações de um autor
  Cenário: 1.1
  Descrição:
  1. O usuário informa o nome ou parte do nome do autor
  2. A aplicação retorna uma lista de autores que possuem o nome informado
  3. O usuário seleciona um autor
  4. A aplicação retorna o nome, a foto, a biografia, uma lista das seções onde o autor publica, e uma lista das matérias publicadas pelo autor, contendo título, data de publicação e resumo.

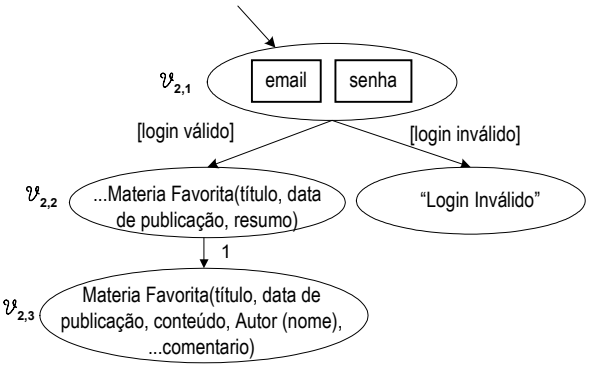
Passo 4: Especificação dos UID's

Deve-se criar um UID para cada caso de uso gerado no passo 3.

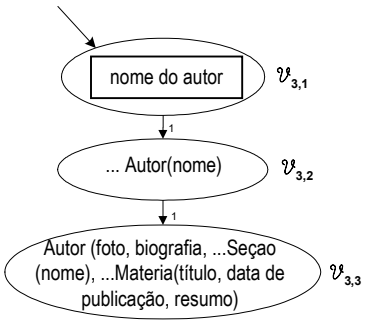
- $\mathcal{U}_1$: Ler matérias de uma seção



- $\mathcal{U}_2$: Consultar matérias favoritas



- $\mathcal{U}_3$: Consultar informações de um autor



A2. Projeto das Visões de Objetos

A atividade Projeto das Visões de Objeto compreende duas sub-atividades: Modelagem dos Tipos das Visões de Objeto e Especificação das Visões de Objeto.

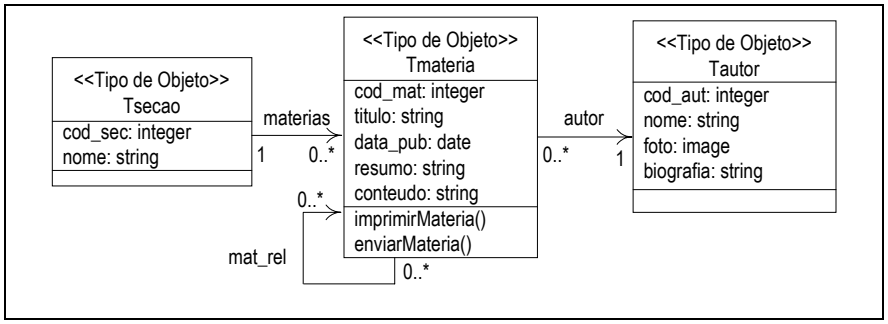
A2.1. Modelagem dos Tipos das Visões de Objetos

Nesta atividade, cada UID obtido na atividade Levantamento de Requisitos é analisado com o objetivo de se definir o diagrama de tipos do UID. Em seguida, os tipos obtidos são integrados de forma a obter os tipos das visões de objeto da aplicação.

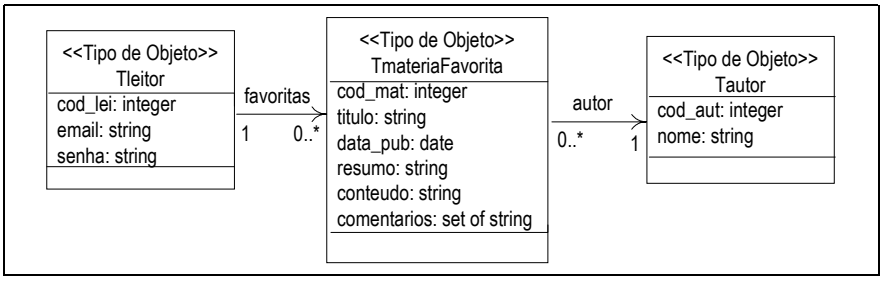
Passo 1: Geração dos Tipos de Objetos das VCN's

Neste passo, os tipos das VCN's são obtidos a partir da sua estrutura.

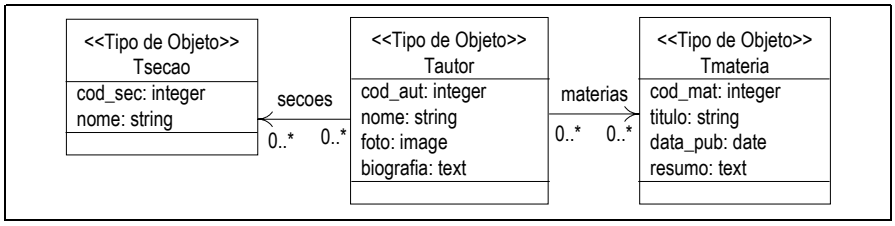
- Diagrama de Tipos das VCN's do UID $\mathcal{U}_1$ (Ler matérias de uma seção)



- Diagrama de Tipos das VCN's do UID $\mathcal{U}_2$ (Consultar matérias favoritas)



- Diagrama de Tipos das VCN's do UID $\mathcal{U}_3$ (Consultar informações de um autor)



Passo 2: Integração dos tipos

Neste passo, os tipos obtidos no passo 1 são integrados para obter os tipos das visões de objetos da aplicação. O diagrama de tipos obtido da integração dos UID's $\mathcal{U}_1$, $\mathcal{U}_2$ e $\mathcal{U}_3$ é mostrado na Figura A.

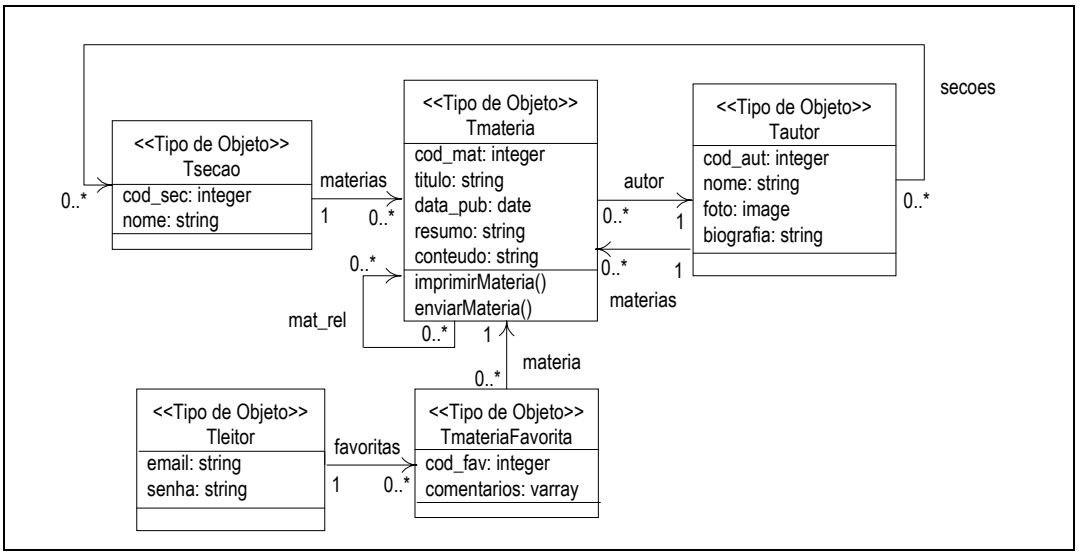


Figura A: Diagrama de tipos integrado

A2.2. Especificação das Visões de Objetos

Neste passo, as visões de objeto da aplicação para os tipos base das VCN's são especificadas conceitualmente por meio de um conjunto de assertivas de correspondência. As AC's são obtidas através do *matching* dos esquemas das visões de objetos com o esquema do banco de dados. A Figura B mostra o esquema das visões de objetos da aplicação Publicações, especificadas sobre o esquema do banco de dados relacional Publicacoes_rel da Figura C. Para visualizar a especificação das visões de objeto, usamos cartões de especificação de visão, apresentados a seguir.

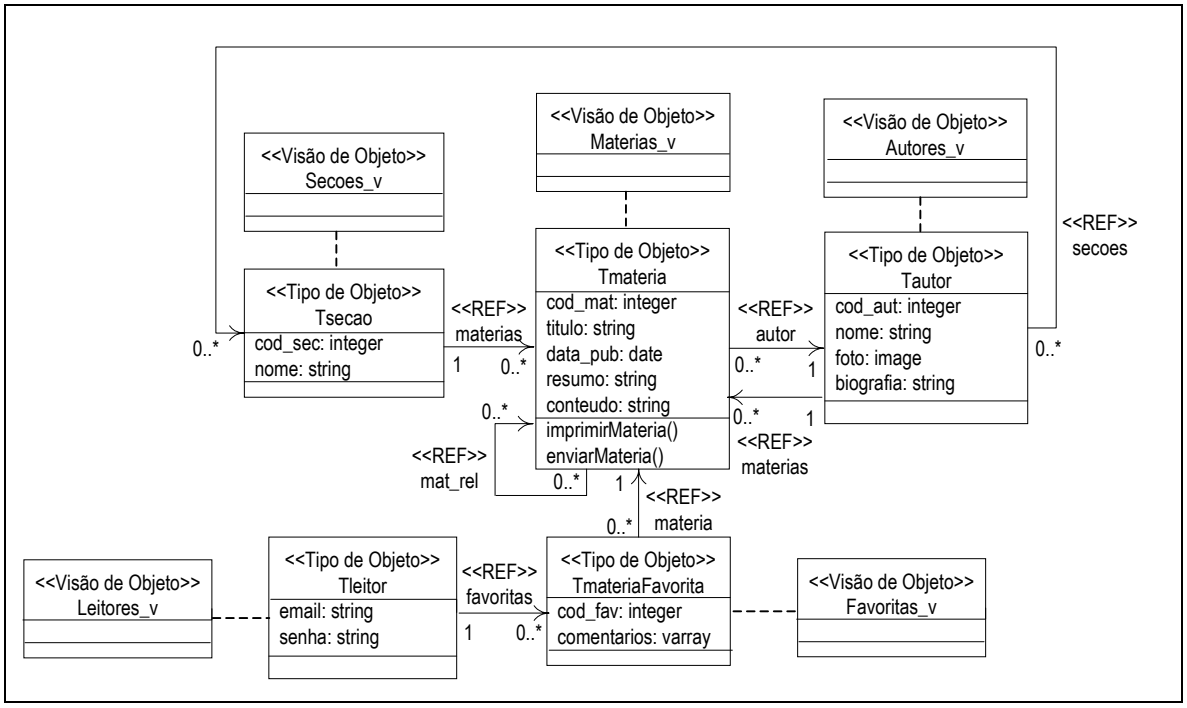


Figura B: Visões de Objetos da aplicação Publicações

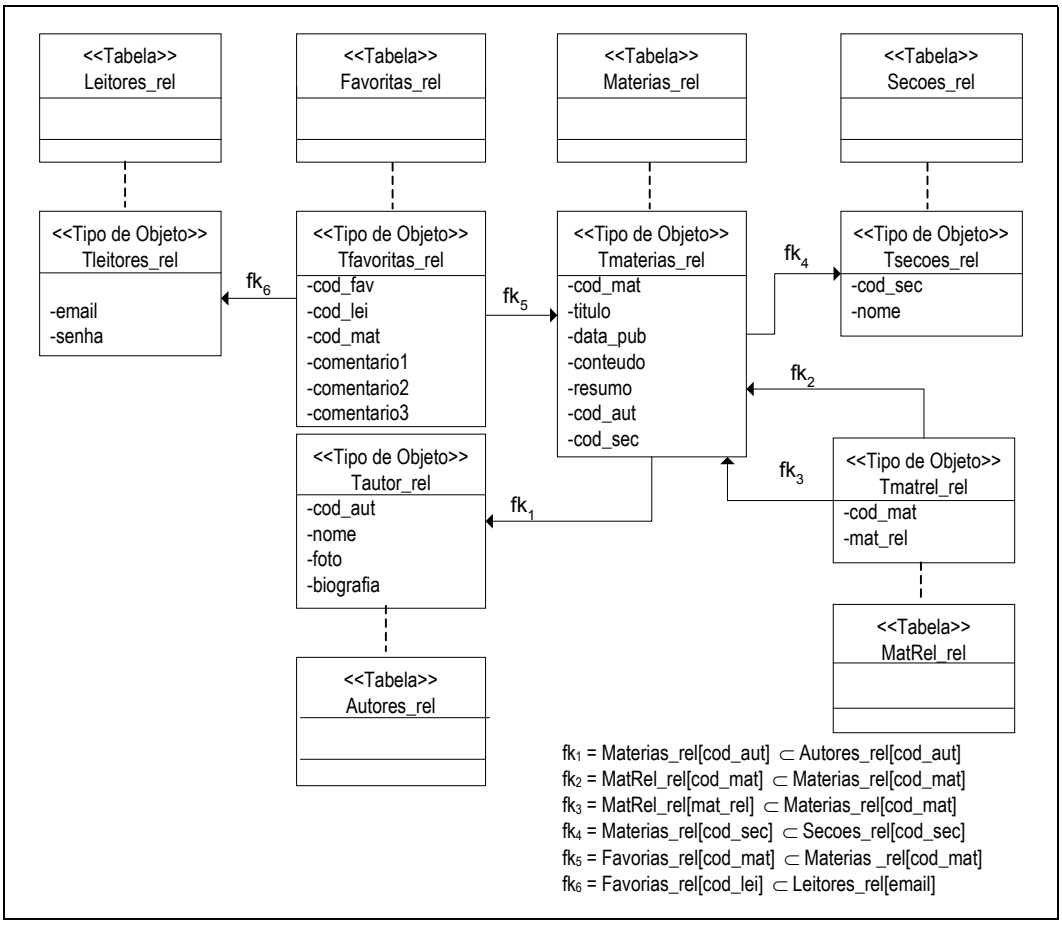


Figura C: Esquema do banco de dados relacional Publicações_rel

❖ Cartões de Especificação das Visões de Objetos da Aplicação

Visão de Objeto: Secoes_v	
Tabela Base: Secoes_rel	**(ACE:** ψ_1: [Secoes_v] ≡ [Secoes_rel]**)** **(ACO:** ψ_2: [Tsecao, {cod_sec}] ≡ [Tsecoes_rel, {cod_sec}]**)**
Atributos: cod_sec: integer nome: string materias: set of REF <<Tmateria>>	**ACC's:** ψ_3: [Tsecao•cod_sec] ≡ [Tsecoes_rel •cod_sec] ψ_4: [Tsecao•nome] ≡ [Tsecoes_rel•nome] ψ_5: [Tsecao•materias] ≡ [Tsecoes_rel•fk_4^{-1}]

Visão de Objeto: Materias_v	
Tabela Base: Materias_rel	**(ACE:** ψ_6: [Materias_v] ≡ [Materias_rel]**)** **(ACO:** ψ_7: [Tmateria, {cod_mat}] ≡ [Tmaterias_rel, {cod_mat}]**)**
Atributos: cod_mat: integer titulo: string data_pub: date resumo: string conteudo: string autor: REF <<Tautor>> mat_rel: set of REF <<Tmateria>>	**ACC's:** ψ_8: [Tmateria•cod_mat] ≡ [Tmaterias_rel•cod_mat] ψ_9: [Tmateria•titulo] ≡ [Tmaterias_rel•titulo] ψ_{10}: [Tmateria•data_pub] ≡ [Tmaterias_rel•data_pub] ψ_{11}: [Tmateria•resumo] ≡ [Tmaterias_rel•resumo] ψ_{12}: [Tmateria•conteudo] ≡ [Tmaterias_rel•conteudo] ψ_{13}: [Tmateria•autor] ≡ [Tmaterias_rel•fk_1] ψ_{14}: [Tmateria•mat_rel] ≡ [Tmaterias_rel•fk_2^{-1}•fk_3]

Visão de Objeto: Autores_v	
Tabela Base: Autores_rel	**(ACE:** ψ_{15}: [Autores_v] ≡ [Autores_rel]**)** **(ACO:** ψ_{16}: [Tautor, {cod_aut}] ≡ [Tautores_rel, {cod_aut}]**)**
Atributos: cod_aut: integer nome: string foto: image biografia: string secoes: set of REF <<Tsecao>> materias: set of REF <<Tmateria>>	**ACC's:** ψ_{17}: [Tautor•cod_aut] ≡ [Tautores_rel•cod_aut] ψ_{18}: [Tautor•nome] ≡ [Tautores_rel•nome] ψ_{19}: [Tautor•foto] ≡ [Tautores_rel•foto] ψ_{20}: [Tautor•biografia] ≡ [Tautores_rel•biografia] ψ_{21}: [Tautor•secao] ≡ [Tautores_rel•fk_1^{-1}•fk_4] ψ_{22}: [Tautor•materias] ≡ [Tautores_rel •fk_1^{-1}]

Visão de Objeto: Leitores_v	
Tabela Base: Leitores_rel	**(ACE:** ψ_{23}: [Leitores_v] ≡ [Leitores_rel]**)** **(ACO:** ψ_{24}: [Tleitor, {email}] ≡ [Tleitores_rel, {email}]**)**
Atributos: email: string senha: string favoritas: set of REF <<TFavorita>>	**ACC's:** ψ_{25}: [Tleitor•email] ≡ [Tleitores_rel•email] ψ_{26}: [Tleitor•senha] ≡ [Tleitores_rel•senha] ψ_{27}: [Tleitor•favoritas] ≡ [Tleitores_rel•fk_6^{-1}]

Visão de Objeto: Favoritas_v	
Tabela Base: Favoritas_rel	**(ACE:** ψ_{28}: [Favoritas_v] ≡ [Favoritas_rel]**)** **(ACO:** ψ_{29}: [TmateriaFavorita, {cod_fav}] ≡ [Tfavoritas_rel, {cod_fav}]**)**
Atributos: cod_fav: integer comentarios: varray of string materia: REF of <<Tmateria>>	**ACC's:** ψ_{30}: [TmateriaFavorita•cod_fav] ≡ [Tfavoritas_rel•cod_fav] ψ_{31}: [TmateriaFavorita•comentarios] ≡ [Tfavoritas_rel, {comentario1, comentario2, comentario3}] ψ_{32}: [TmateriaFavorita•materia] ≡ [Tfavoritas_rel•fk_5]

A3. Projeto das VCN's

Nesta atividade, as VCN's são definidas sobre sua visão de objeto base. Para visualização da especificação conceitual das VCN's, usamos cartões de especificação. A seguir, apresentamos os cartões das VNC's dos UID's $\mathcal{U}_1$, $\mathcal{U}_2$ e $\mathcal{U}_3$.

❖ Cartões de Especificação das VCN's do UID $\mathcal{U}_1$(Ler matérias de uma seção)

VCN: $\mathcal{V}_{1,1}$		
Parâmetros:		
Visão de Objeto Base: Secoes_v	**Condição de Seleção:**	**(ACE**: $\psi_1^{[V1,1]}$: [$V_{1,1}$] ≡ Secoes_v**)**
Atributos: cod_sec: integer ↘nome: string	**Assertivas de Correspondência:** $\psi_2^{[V1,1]}$: [$T_{[V1,1]}$•cod_sec] ≡ [Tsecao•cod_sec] $\psi_3^{[V1,1]}$: [$T_{[V1,1]}$•nome] ≡ [Tsecao•nome]	
Elos: materias: {**Destino**: $V_{1,2}$; **Parâmetros**:{self•cod_sec}}		
Operações:		
Ordenação: nome		

VCN: $\mathcal{V}_{1,2}$		
Parâmetros: param: integer		
Visão de Objeto Base: Secoes_v	**Condição de Seleção:** cod_sec = param	**(ACE**: $\psi_1^{[V1,2]}$: [$V_{1,2}$] ≡ [Secoes_v[cod_sec = param]]**)**
Atributos: materias: set of <$T_{[materia]}$> cod_mat: integer data_publicacao: string ↘titulo: string resumo: string	**Assertivas de Correspondência:** $\psi_2^{[V1,2]}$: [$T_{[V1,2]}$•materias] ≡ [Tsecao•materias] $\psi_3^{[V1,2]}$: [$T_{[materia]}$•cod_mat] ≡ [Tmateria• cod_mat] $\psi_4^{[V1,2]}$: [$T_{[materia]}$•data_publicacao] ≡ [Tmateria•data_pub] $\psi_5^{[V1,2]}$: [$T_{[materia]}$•titulo] ≡ [Tmateria•titulo] $\psi_6^{[V1,2]}$: [$T_{[materia]}$•resumo] ≡ [Tmateria•resumo]	
Elos: inf_materia: {**Destino**: $V_{1,3}$; **Parâmetros**:{self•mat•cod_mat|mat ∈ materias]}		

Operações:
Ordenação: materias•titulo

VCN: $\mathcal{V}_{1,3}$	
Parâmetros: param: integer	
Visão de Objeto Base: Materias_v	**Condição de Seleção:** cod_mat = param **(ACE**: $\psi_1^{[V1,3]}$: $[V_{1,3}] \equiv$ [Materias_v[cod_mat = param]]**)**
Atributos:	**Assertivas de Correspondência:**
titulo: string	$\psi_2^{[V1,3]}$: $[T_{[V1,3]}$•titulo] ≡ [Tmateria•titulo]
data_de_publicacao: string	$\psi_3^{[V1,3]}$: $[T_{[V1,3]}$•data_de_publicacao] ≡ [Tmateria•data_pub]
conteudo: string	$\psi_4^{[V1,3]}$: $[T_{[V1,3]}$•conteudo] ≡ [Tmateria•conteudo]
autor: <$T_{[autor]}$>	$\psi_5^{[V1,3]}$: $[T_{[V1,3]}$•autor] ≡ [Tmateria•autor]
cod_aut: integer	$\psi_6^{[V1,3]}$: $[T_{[autor]}$•cod_aut] ≡ [Tautor •cod_aut]
↘nome: string	$\psi_7^{[V1,3]}$: $[T_{[autor]}$•nome] ≡ [Tautor•nome]
mat_rel: set of <$T_{[mat\_rel]}$>	$\psi_8^{[V1,3]}$: $[T_{[V1,3]}$•mat_rel] ≡ [Tmateria•mat_rel]
cod_mat: integer	$\psi_9^{[V1,3]}$: $[T_{[mat\_rel]}$•cod_mat] ≡ [Tmateria•cod_mat]
↘titulo: string	$\psi_{10}^{[V1,3]}$: $[T_{[mat\_rel]}$•titulo] ≡ [Tmateria•titulo]
data_pub: string	$\psi_{11}^{[V1,3]}$: $[T_{[mat\_rel]}$•data_pub] ≡ [Tmateria•data_pub]
Elos: inf_autor: {**Destino**: $V_{1,4}$; **Parâmetros**:{self•autor•cod_aut}} inf_mat_rel:{**Destino**:$V_{1,3}$; **Parâmetros**:{self•m•titulo | m ∈ mat_rel}}	
Operações: imprimirMateria() enviarMateria()	
Ordenação:	

VCN: $\mathcal{V}_{1,4}$	
Parâmetros: param: integer	
Visão de Objeto Base: Autores_v	**Condição de Seleção:** cod_aut = param **(ACE**: $\psi_1^{[V1,4]}$: $[V_{1,4}] \equiv$ [Autores_v[cod_aut = param]]**)**
Atributos:	**Assertivas de Correspondência:**
nome: string	$\psi_2^{[V1,4]}$: $[T_{[V1,4]}$•nome] ≡ [Tautor•nome]
foto: string	$\psi_3^{[V1,4]}$: $[T_{[V1,4]}$•foto] ≡ [Tautor•foto]
biografia: string	$\psi_4^{[V1,4]}$: $[T_{[V1,4]}$•biografia] ≡ [Tautor•biografia]
Elos:	
Operações:	
Ordenação:	

❖ Cartões de Especificação das VCN's do UID $\mathcal{U}_2$ (Consultar matérias favoritas)

VCN: $\mathcal{V}_{2,2}$	
Parâmetros: param: string	
Visão de Objeto Base: Leitores_v	**Condição de Seleção:** email = param **(ACE**: $\psi_1^{[V2,2]}$: $[V_{2,2}] \equiv$ [Leitores_v[email = param]])
Atributos:	**Assertivas de Correspondência:**
materias_favoritas: set of <$T_{[materiaFavorita]}$>	$\psi_2^{[V2,2]}$: $[T_{[V2,2]} \bullet$materias_favoritas$] \equiv$ [Tleitor•favoritas]
cod_fav: integer	$\psi_3^{[V2,2]}$: $[T_{[materiaFavorita]} \bullet$cod_fav$] \equiv$ [Tfavorita•cod_fav]
materia: <$T_{[materia]}$>	$\psi_4^{[V2,2]}$: $[T_{[materiaFavorita]} \bullet$materia$] \equiv$ [Tfavorita•materia]
data_publicacao: string	$\psi_5^{[V2,2]}$: $[T_{[materia]} \bullet$data_publicacao$] \equiv$ [Tmateria•data_pub]
↘titulo: string	$\psi_6^{[V2,2]}$: $[T_{[materia]} \bullet$titulo$] \equiv$ [Tmateria•titulo]
resumo: string	$\psi_7^{[V2,2]}$: $[T_{[materia]} \bullet$resumo$] \equiv$ [Tmateria•resumo]
Elos: inf_mat: {**Destino**: $V_{2,3}$; **Parâmetros**:{self•m•cod_fav | m ∈ materias_favoritas}}	
Operações:	
Ordenação: materias_favoritas•titulo	

VCN: $\mathcal{V}_{2,3}$	
Parâmetros: param: integer	
Visão de Objeto Base: Favoritas_v	**Condição de Seleção:** cod_fav = param **(ACE**: $\psi_1^{[V2,3]}$: $[V_{2,3}] \equiv$ [Favoritas_v[cod_fav = param]])
Atributos:	**Assertivas de Correspondência:**
materia: <$T_{[materia]}$>	$\psi_2^{[V2,3]}$: $[T_{[V2,3]} \bullet$materia$] \equiv$ [Tfavorita•materia]
titulo: string	$\psi_3^{[V2,3]}$: $[T_{[materia]} \bullet$titulo$] \equiv$ [Tfavorita•materia•titulo]
data_publicacao: string	$\psi_4^{[V2,3]}$: $[T_{[materia]} \bullet$data_publicacao$] \equiv$ [Tfavorita•materia•data_pub]
conteudo: string	$\psi_5^{[V2,3]}$: $[T_{[materia]} \bullet$conteudo$] \equiv$ [Tfavorita•materia•conteudo]
autor: string	$\psi_6^{[V2,3]}$: $[T_{[materia]} \bullet$autor$] \equiv$ [Tfavorita•materia•autor•nome]
comentarios: set of <string>	$\psi_7^{[V2,3]}$: $[T_{[V2,3]} \bullet$comentarios$] \equiv$ [Tfavorita•comentarios]
Elos:	
Operações:	
Ordenação:	

❖ Cartões de Especificação das VCN's do UID $\mathcal{U}_3$ (Consultar informações de um autor)

VCN: $\mathcal{V}_{3,2}$	
Parâmetros: param: string	
Visão de Objeto Base: Autores_v	**Condição de Seleção:** nome = param **(ACE**: $\psi_1^{[V3,2]}$: $[V_{3,2}] \equiv$ [Autores_v [nome = param]])
Atributos:	**Assertivas de Correspondência:**
cod_aut: integer	$\psi_2^{[V3,2]}$: $[T_{[V3,2]} \bullet$cod_aut$] \equiv$ [Tautor•cod_aut]
↘nome: string	$\psi_3^{[V3,2]}$: $[T_{[V3,2]} \bullet$nome$] \equiv$ [Tautor•nome]
Elos: inf_aut: {**Destino**: $V_{3,3}$; **Parâmetros**:{self•cod_aut}}	
Operações:	
Ordenação: nome	

VCN: $\mathcal{V}_{3,3}$		
Parâmetros: param: string		
Visão de Objeto Base: Autores_v	**Condição de Seleção:** cod_aut = param	**(ACE**: $\psi_1^{[V3,3]}$: $[V_{3,3}] \equiv$ [Autores_v[cod_aut = param]]**)**
Atributos:	**Assertivas de Correspondência:**	
foto: string	$\psi_2^{[V3,3]}$: $[T_{[V3,3]} \bullet$foto$] \equiv$ [Tautor•foto]	
biografia: string	$\psi_3^{[V3,3]}$: $[T_{[V3,3]} \bullet$biografia$] \equiv$ [Tautor•biografia]	
seções: set of <string>	$\psi_4^{[V3,3]}$: $[T_{[V3,3]} \bullet$secoes$] \equiv$ [Tautor•secoes•nome]	
matérias: set of <$T_{[materia]}$>	$\psi_5^{[V3,3]}$: $[T_{[V3,3]} \bullet$materias$] \equiv$ [Tautor•materias]	
titulo: string	$\psi_6^{[V3,3]}$: $[T_{[materia]} \bullet$titulo$] \equiv$ [Tmateria •titulo]	
data_publicacao: string	$\psi_7^{[V3,3]}$: $[T_{[materia]} \bullet$data_publicacao$] \equiv$ [Tmateria•data_pub]	
resumo: string	$\psi_8^{[V3,3]}$: $[T_{[materia]} \bullet$resumo$] \equiv$ [Tmateria•resumo]	
Elos:		
Operações:		
Ordenação:		

A4. Implementação das VCN's

Neste trabalho, consideramos duas abordagens para implementação das VCN's: como visões de objeto e como visões XML. A seguir, mostramos como é feita a implementação das VCN's utilizando cada uma das abordagens citadas.

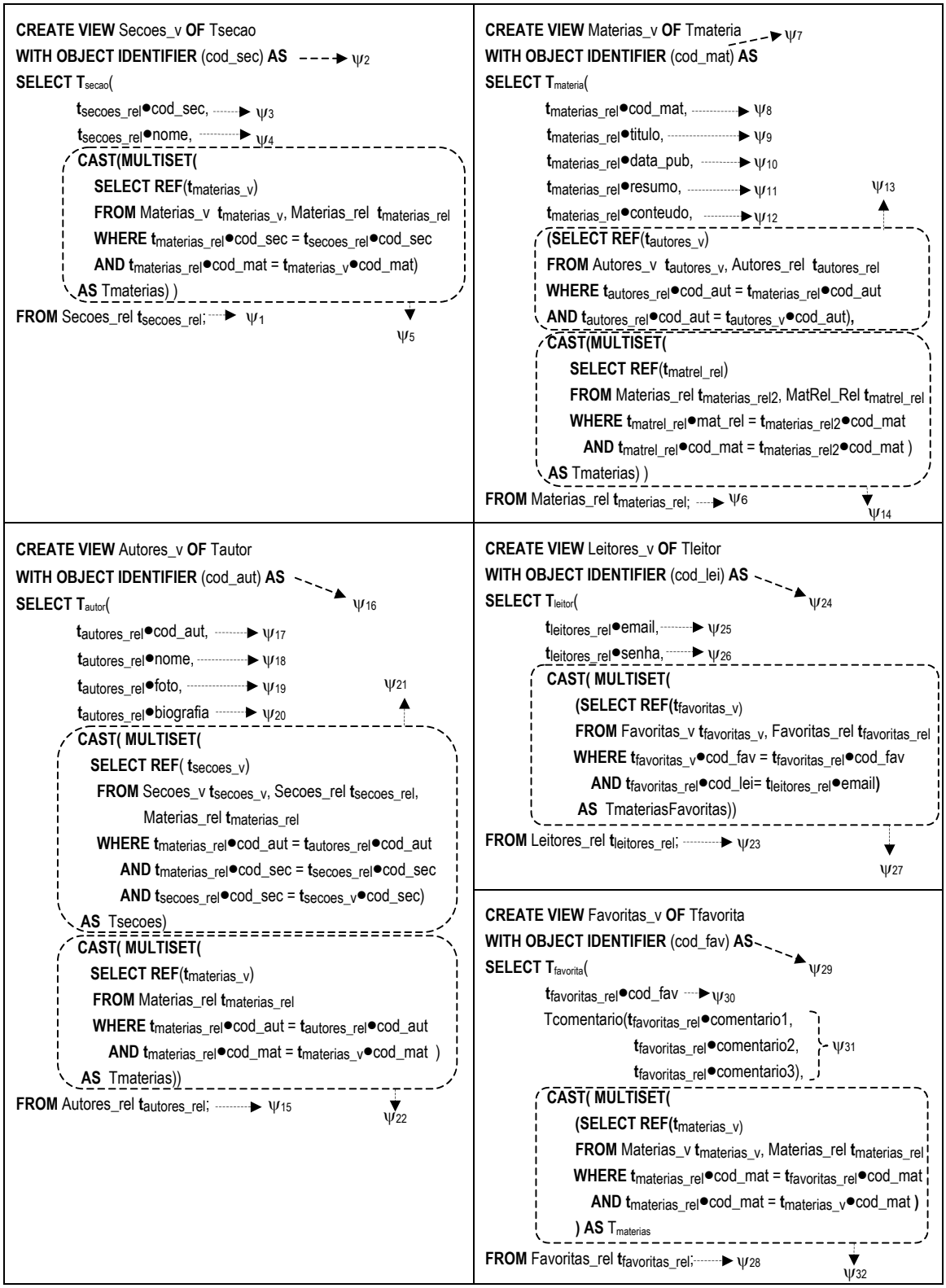
A4.1. Implementação como Visões de Objeto

Nesta abordagem, a definição da VCN consiste de uma consulta SQL que, quando executada, sintetiza os objetos da VCN a partir dos dados no banco de dados. As consultas SQL que definem as VCN's são armazenadas em procedimentos armazenados. A implementação das VCN's pode ser de duas formas: implementação em 3 camadas, onde as consultas SQL são definidas sobre o esquema da visão de objeto base, e implementação em 2 camadas, onde as consultas SQL são definidas sobre o esquema do banco de dados.

A4.1.1. Implementação em 3 camadas

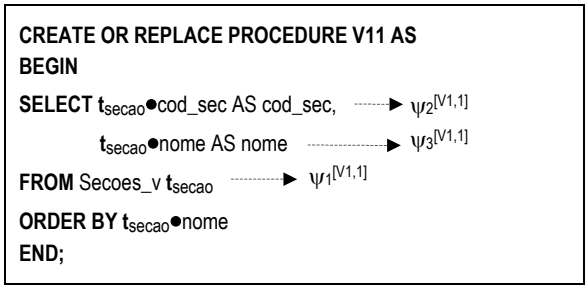
Neste caso, o processo de implementação das VCN's consiste de dois passos, apresentados a seguir.

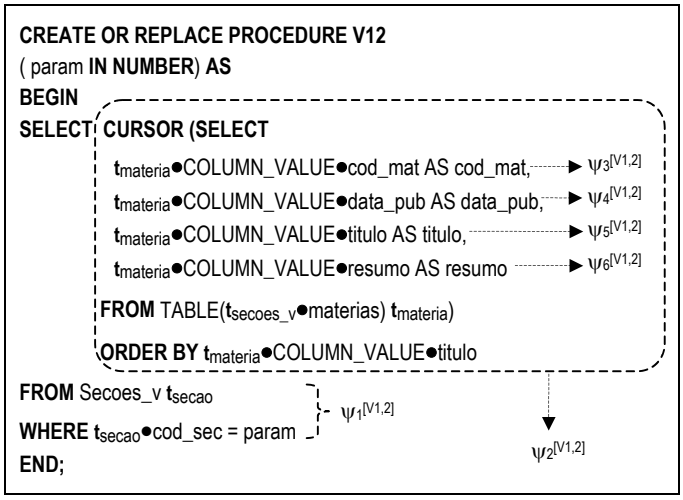
Passo 1: Criar as Visões de Objetos

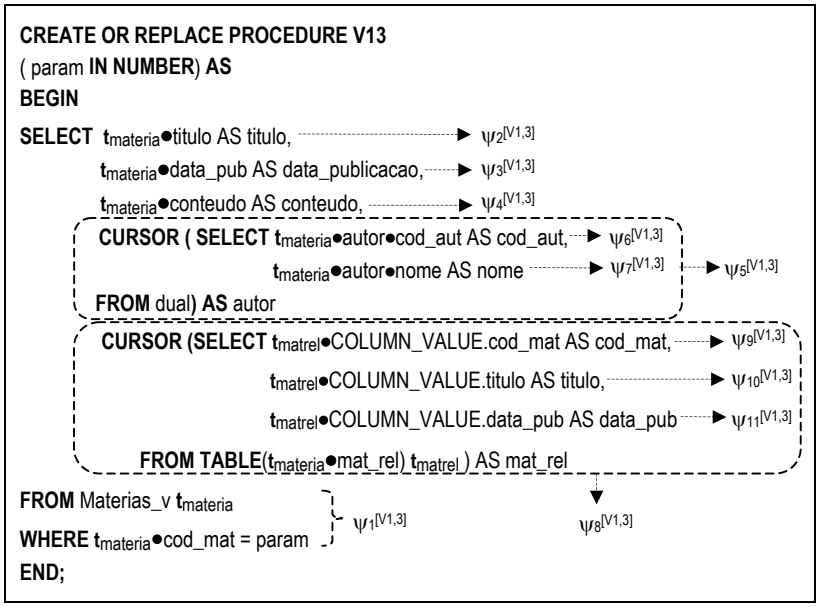


Passo 2: Gerar definição das VCN sobre esquema da visão de objeto base

❖ VCN's do UID $\mathcal{U}_1$







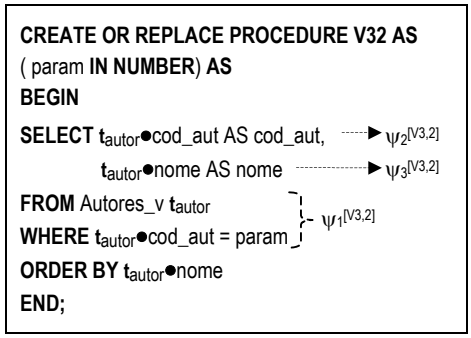
```
CREATE OR REPLACE PROCEDURE V14
( param IN NUMBER ) AS
BEGIN
SELECT t_autor●nome AS nome,        ──▶ ψ2[V1,4]
       t_autor●foto AS foto,        ──▶ ψ3[V1,4]
       t_autor●biografia AS biografia ──▶ ψ4[V1,4]
FROM Autores_v t_autor                  } ψ1[V14]
WHERE t_autor●cod_aut = param           }
END;
```

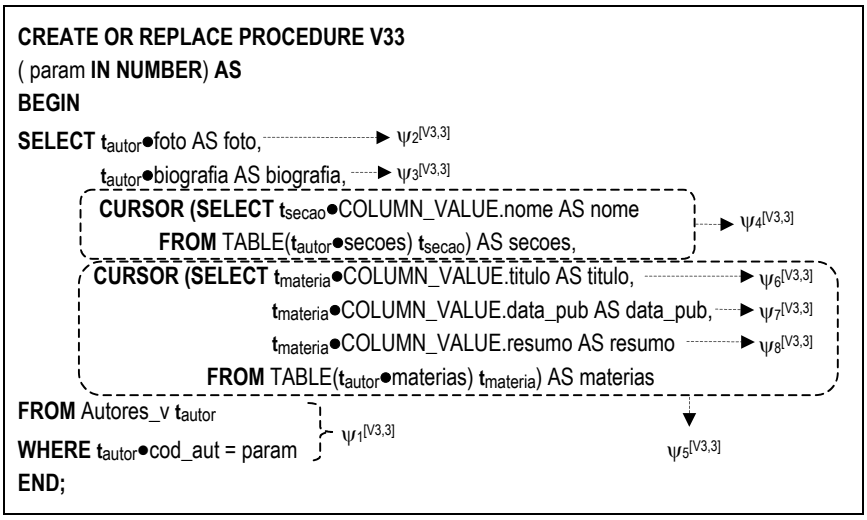
❖ VCN's do UID $\mathcal{U}_2$

```
CREATE OR REPLACE PROCEDURE V22
( param IN VARCHAR) AS
BEGIN
SELECT CURSOR (SELECT
         t_favorita●COLUMN_VALUE●cod_fav AS cod_fav,        ──▶ ψ3[V2,2]
         CURSOR (SELECT
                   t_favorita●COLUMN_VALUE●materia●data_pub AS data_pub,
                   t_favorita●COLUMN_VALUE●materia●titulo AS titulo,        ──▶ ψ4[V2.2]
                   t_favorita●COLUMN_VALUE●materia●resumo AS resumo
                 FROM DUAL ) AS materia
         FROM TABLE(t_leitor●favoritas) t_favorita) AS materias_favoritas
         ORDER BY t_favorita●COLUMN_VALUE●materia●titulo
FROM Leitores_v t_leitor                  } ψ1[V2,2]
WHERE t_leitor●email = param              }
END;                                                  ▼ ψ2[V2.2]
```

```
CREATE OR REPLACE PROCEDURE V23
( param IN NUMBER) AS
BEGIN
SELECT CURSOR( SELECT
                 t_favorita●materia●titulo AS titulo,        ──▶ ψ3[V2,3]
                 t_favorita●materia●data_pub AS data_publicacao,  ──▶ ψ4[V2,3]
                 t_favorita●materia●conteudo AS conteúdo,    ──▶ ψ5[V2,3]   ──▶ ψ2[V2.2]
                 t_favorita●materia●autor●nome AS autor,     ──▶ ψ6[V2,3]
                 FROM DUAL) AS materia,
       t_favorita●comentarios AS comentarios                 ──▶ ψ7[V2,3]
FROM Favoritas_v t_favorita               } ψ1[V2,3]
WHERE t_favorita●cod_fav = param          }
END;
```

❖ VCN's do UID $\mathcal{U}_3$





A4.1.2. Implementação em 2 camadas

Neste caso, as VCN's são definidas sobre o esquema do banco de dados. Considere o banco de dados relacional Publicações_rel apresentado na Figura C. O processo de implementação das VCN's consiste de dois passos, apresentados a seguir.

Passo 1: Geração das AC's da VCN com o esquema do banco de dados

❖ Especificação das VCN's do UID $\mathcal{U}_1$ com o banco de dados relacional

♦ Cartão de Especificação da VCN $\mathcal{V}_{1,1}$

VCN: $\mathcal{V}_{1,1}$	
Parâmetros:	
Tabela Base: Secoes_rel	**Condição de Seleção:** **(ACE**: $\delta_1^{[V1,1]}$: $[V_{1,1}] \equiv$ [Secoes_rel] (de $\psi_1^{[V1,1]}$ e ψ_1)**)**
Atributos: cod_sec: integer ↘nome: string	**Assertivas de Correspondência:** $\delta_2^{[V1,1]}$: $[T_{[V1,1]} \bullet \text{cod\_sec}] \equiv [T_{\text{secoes\_rel}} \bullet \text{cod\_sec}]$ (de $\psi_2^{[V1,1]}$ e ψ_3) $\delta_3^{[V1,1]}$: $[T_{[V1,1]} \bullet \text{nome}] \equiv [T_{\text{secoes\_rel}} \bullet \text{nome}]$ (de $\psi_3^{[V1,1]}$ e ψ_4)
Elos: materias: {**Destino**: $V_{1,2}$; **Parâmetros**:{self•cod_sec}}	
Operações:	
Ordenação: nome	

♦ Cartão de Especificação da VCN $\mathcal{V}_{1,2}$

VCN: $\mathcal{V}_{1,2}$	
Parâmetros: param: integer	
Tabela Base: Secoes_rel	**Condição de Seleção:** cod_sec = param **(ACE**: $\delta_1^{[V1,2]}$: $[V_{1,2}] \equiv$ [Secoes_rel[cod_sec = param]] (de $\psi_1^{[V1,2]}$ e ψ_1)**)**
Atributos: materias: set of <$T_{[materia]}$> cod_mat: integer data_de_publicacao: string ↘titulo: string resumo: string	**Assertivas de Correspondência:** $\delta_2^{[V1,2]}$: $[T_{[V1,2]} \bullet \text{materias}] \equiv [T_{\text{secoes\_rel}} \bullet fk_4^{-1}]$ (de $\psi_2^{[V1,2]}$ e ψ_5) $\delta_3^{[V1,2]}$: $[T_{[materia]} \bullet \text{cod\_mat}] \equiv [T_{\text{materias\_rel}} \bullet \text{cod\_mat}]$ (de $\psi_3^{[V1,2]}$ e ψ_8) $\delta_4^{[V1,2]}$: $[T_{[materia]} \bullet \text{data\_pub}] \equiv [T_{\text{materias\_rel}} \bullet \text{data\_pub}]$ (de $\psi_4^{[V1,2]}$ e ψ_{10}) $\delta_5^{[V1,2]}$: $[T_{[materia]} \bullet \text{titulo}] \equiv [T_{\text{materias\_rel}} \bullet \text{titulo}]$ (de $\psi_5^{[V1,2]}$ e ψ_9) $\delta_6^{[V1,2]}$: $[T_{[materia]} \bullet \text{resumo}] \equiv [T_{\text{materias\_rel}} \bullet \text{nome}]$ (de $\psi_6^{[V1,2]}$ e ψ_{11})
Elos: inf_materia: {**Destino**: $V_{1,3}$; **Parâmetros**:{self•mat•cod_mat|mat ∈ materias]}	
Operações:	
Ordenação: materias•titulo	

♦ Cartão de Especificação da VCN $\mathcal{V}_{1,3}$

VCN: $\mathcal{V}_{1,3}$	
Parâmetros: param: integer	
Tabela Base: Materias_rel	**Condição de Seleção:** cod_mat = param **(ACE**: $\delta_1^{[V1,3]}$: $[V_{1,3}] \equiv$ [Materias_rel[cod_mat = param]] (de $\psi_1^{[V1,3]}$ e ψ_2)**)**
Atributos: titulo: string data_de_publicacao: string	**Assertivas de Correspondência:** $\delta_2^{[V1,3]}$: $[T_{[V1,3]} \bullet \text{titulo}] \equiv [T_{\text{materias\_rel}} \bullet \text{titulo}]$ (de $\psi_2^{[V1,3]}$ e ψ_9) $\delta_3^{[V1,3]}$: $[T_{[V1,3]} \bullet \text{data\_pub}] \equiv [T_{\text{materias\_rel}} \bullet \text{data\_pub}]$ (de $\psi_3^{[V1,3]}$ e ψ_{10})

conteudo: string	$\delta_4^{[V1,3]}$: $[T_{[V1,3]} \bullet \text{conteudo}] \equiv [T_{\text{materias\_rel}} \bullet \text{conteudo}]$ (de $\psi_4{}^{[V1,3]}$ e ψ_{12})
autor: <$T_{[autor]}$>	$\delta_5^{[V1,3]}$: $[T_{[V1,3]} \bullet \text{autor}] \equiv [T_{\text{materias\_rel}} \bullet fk_1]$ (de $\psi_5^{[V1,3]}$ e ψ_{13})
cod_aut: integer	$\delta_6^{[V1,3]}$: $[T_{[\text{autores\_rel}]} \bullet \text{cod\_aut}] \equiv [T_{\text{autores\_rel}} \bullet \text{cod\_aut}]$ (de $\psi_6^{[V1,3]}$ e ψ_{17})
↘nome: string	$\delta_7^{[V1,3]}$: $[T_{[\text{autores\_rel}]} \bullet \text{nome}] \equiv [T_{\text{autores\_rel}} \bullet \text{nome}]$ (de $\psi_7^{[V1,3]}$ e ψ_{18})
mat_rel: set of <$T_{[mat\_rel]}$>	$\delta_8^{[V1,3]}$: $[T_{[V1,3]} \bullet \text{mat\_rel}] \equiv [T_{\text{materias\_rel}} \bullet fk_2^{-1} \bullet fk_3]$ (de $\psi_8^{[V1,3]}$ e ψ_{14})
cod_mat: integer	$\delta_9^{[V1,3]}$: $[T_{[mat\_rel]} \bullet \text{cod\_mat}] \equiv [T_{\text{materias\_rel}} \bullet \text{cod\_mat}]$ (de $\psi_9^{[V1,3]}$ e ψ_8)
↘titulo: string	$\delta_{10}^{[V1,3]}$: $[T_{[mat\_rel]} \bullet \text{titulo}] \equiv [T_{\text{materias\_rel}} \bullet \text{titulo}]$ (de $\psi_{10}^{[V1,3]}$ e ψ_9)
data_de_publicacao: string	$\delta_{11}^{[V1,3]}$: $[T_{[mat\_rel]} \bullet \text{data\_publicacao}] \equiv [T_{\text{materias\_rel}} \bullet \text{data\_pub}]$ (de $\psi_{11}^{[V1,3]}$ e ψ_{10})
Elos:	
inf_autor: {**Destino**: $V_{1,4}$; **Parâmetros**:{self•autor•cod_aut}}	
inf_mat_rel:{**Destino**:$V_{1,3}$; **Parâmetros**:{self•m•titulo | m ∈ mat_rel}}	
Operações:	
imprimirMateria()	
enviarMateria()	
Ordenação:	

- Cartão de Especificação da VCN $\mathcal{V}_{1,4}$

VCN: $\mathcal{V}_{1,4}$	
Parâmetros: param: integer	
Tabela Base: Autores_rel	**Condição de Seleção:** cod_aut = param **(ACE**: $\delta^1_{[V1,4]}$: $[V_{1,4}] \equiv$ [Autores_rel[cod_aut = param]] (de $\psi^1_{[V1,4]}$ e ψ^3))
Atributos:	**Assertivas de Correspondência:**
nome: string	$\delta_2^{[V1,4]}$: $[T_{[V1,3]} \bullet \text{nome}] \equiv [T_{\text{autores\_rel}} \bullet \text{nome}]$ (de $\psi_2^{[V1,4]}$ e ψ_{18})
foto: string	$\delta_3^{[V1,4]}$: $[T_{[V1,3]} \bullet \text{foto}] \equiv [T_{\text{autores\_rel}} \bullet \text{foto}]$ (de $\psi_3^{[V1,4]}$ e ψ_{19})
biografia: string	$\delta_4^{[V1,4]}$: $[T_{[V1,3]} \bullet \text{biografia}] \equiv [T_{\text{autores\_rel}} \bullet \text{biografia}]$ (de $\psi_4^{[V1,4]}$ e ψ_{20})
Elos:	
Operações:	
Ordenação:	

- Especificação das VCN's do UID $\mathcal{U}_2$ com o banco de dados relacional

  - Cartão de Especificação da VCN $\mathcal{V}_{2,2}$

VCN: $\mathcal{V}_{2,2}$	
Parâmetros: param: string	
Tabela Base: Leitores_rel	**Condição de Seleção:** email = param **(ACE**: $\delta_1^{[V2,2]}$: $[V_{2,2}] \equiv$ [Leitores_rel[email = param]] (de $\psi_1^{[V2,2]}$ e ψ_{23}))
Atributos:	**Assertivas de Correspondência:**
materias_favoritas: set of <$T_{[materiaFavorita]}$>	$\delta_2^{[V2,2]}$: $[T_{[V2,2]} \bullet \text{materias\_favoritas}] \equiv [T_{\text{leitores\_rel}} \bullet fk_6^{-1}]$ (de $\psi_2^{[V2,2]}$, ψ_{27})
cod_fav: integer	$\delta_3^{[V2,2]}$: $[T_{[\text{materiaFavorita}]} \bullet \text{cod\_fav}] \equiv [T_{\text{favoritas\_rel}} \bullet \text{cod\_fav}]$ (de $\psi_3^{[V2,2]}$ e ψ_{30})
materia: <$T_{[materia]}$>	$\delta_4^{[V2,2]}$: $[T_{[\text{materiaFavorita}]} \bullet \text{materia}] \equiv [T_{\text{favoritas\_rel}} \bullet fk_5]$ (de $\psi_4^{[V2,2]}$ e ψ_{32})
data_publicacao: string	$\delta_5^{[V2,2]}$: $[T_{[materia]} \bullet \text{data\_publicacao}] \equiv [T_{\text{materias\_rel}} \bullet \text{data\_pub}]$ (de $\psi_5^{[V2,2]}$ e ψ_{10})
↘titulo: string	$\delta_6^{[V2,2]}$: $[T_{[materia]} \bullet \text{titulo}] \equiv [T_{\text{materias\_rel}} \bullet \text{titulo}]$ (de $\psi_6^{[V2,2]}$ e ψ_9)
resumo: string	$\delta_7^{[V2,2]}$: $[T_{[materia]} \bullet \text{resumo}] \equiv [T_{\text{materias\_rel}} \bullet \text{resumo}]$ (de $\psi_7^{[V2,2]}$ e ψ_{11})
Elos:	
inf_mat: {**Destino**: $V_{2,3}$; **Parâmetros**:{self•m•cod_fav | m ∈ materias_favoritas}}	

Operações:
Ordenação: materias_favoritas•materia•titulo

- Cartão de Especificação da VCN $\mathcal{V}_{2,3}$

VCN: $\mathcal{V}_{2,3}$	
Parâmetros: param: integer	
Tabela Base: Favoritas_rel	**Condição de Seleção:** cod_fav = param **(ACE**: $\delta_1^{[V2,3]}$: $[V_{2,3}] \equiv$ [Favoritas_rel[cod_fav = param]] (de $\psi_1^{[V2,3]}$ e ψ_{23})**)**
Atributos: materia: <$T_{[materia]}$> titulo: string data_publicacao: string conteudo: string autor: string comentarios: set of <string>	**Assertivas de Correspondência:** $\delta_2^{[V2,3]}$: $[T_{[V2,3]}$•materia$] \equiv [T_{favoritas_rel}$•$fk_5]$ (de $\psi_2^{[V2,3]}$ e ψ_{32}) $\delta_3^{[V2,3]}$: $[T_{[materia]}$•titulo$] \equiv [T_{materias_rel}$•titulo] (de $\psi_3^{[V2,3]}$ e ψ_9) $\delta_4^{[V2,3]}$: $[T_{[materia]}$•data_publicacao$] \equiv [T_{materias_rel}$•data] (de $\psi_4^{[V2,3]}$ e ψ_{10}) $\delta_5^{[V2,3]}$: $[T_{[materia]}$•conteudo$] \equiv [T_{materias_rel}$•conteudo] (de $\psi_5^{[V2,3]}$ e ψ_{12}) $\delta_6^{[V2,3]}$: $[T_{[materia]}$•autor$] \equiv [T_{materias_rel}$•autor•nome] (de $\psi_6^{[V2,3]}$, ψ_{13}, ψ_{18} e ψ_{18}) $\delta_7^{[V2,3]}$: $[T_{[V2,3]}$•comentarios$] \equiv [T_{favoritas_rel}$, {comentario1, comentario2, comentario3}] (de $\psi_7^{[V2,3]}$ e ψ_{31})
Elos:	
Operações:	
Ordenação:	

- Especificação das VCN's do UID $\mathcal{U}_3$ com o banco de dados relacional

  - Cartão de Especificação da VCN $\mathcal{V}_{3,2}$

VCN: $\mathcal{V}_{3,2}$	
Parâmetros: param: string	
Tabela Base: Autores_rel	**Condição de Seleção:** nome LIKE param **(ACE**: $\delta_1^{[V3,2]}$: $[V_{3,2}] \equiv$ [Autores_rel[nome = param]] (de $\psi_1^{[V3,2]}$ e ψ_{15})**)**
Atributos: cod_aut: integer ↘nome: string	**Assertivas de Correspondência:** $\delta_2^{[V3,2]}$: $[T_{[V3,2]}$•cod_aut$] \equiv [T_{autores_rel}$•cod_aut] (de $\psi_2^{[V1,4]}$ e ψ_{17}) $\delta_3^{[V3,2]}$: $[T_{[V3,2]}$•nome$] \equiv [T_{autores_rel}$•nome] (de $\psi_3^{[V1,4]}$ e ψ_{18})
Elos: inf_aut: {**Destino**: $V_{3,3}$; **Parâmetros**:{self•cod_aut}}	
Operações:	
Ordenação: nome	

- Cartão de Especificação da VCN $\mathcal{V}_{3,3}$

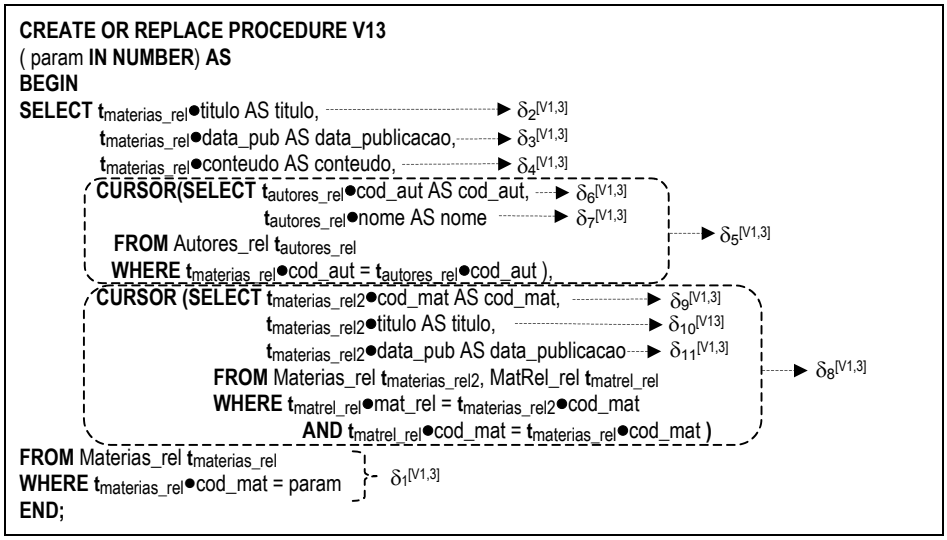
VCN: $\mathcal{V}_{3,3}$	
Parâmetros: param: string	
Tabela Base: Autores_rel	**Condição de Seleção:** cod_aut = param **(ACE**: $\delta_1^{[V3,3]}$: $[V_{3,3}] \equiv$ [Autores_rel[cod_aut = param]] (de $\psi_1^{[V3,3]}$ e ψ_{15})**)**
Atributos:	**Assertivas de Correspondência:**
foto: string	$\delta_2^{[V3,3]}$: $[T_{[V3,3]} \bullet \text{foto}] \equiv T_{\text{autores\_rel}} \bullet \text{foto}$ (de $\psi_2^{[V1,4]}$ e ψ_{19})
biografia: string	$\delta_3^{[V3,3]}$: $[T_{[V3,3]} \bullet \text{biografia}] \equiv T_{\text{autores\_rel}} \bullet \text{biografia}$ (de $\psi_3^{[V1,4]}$ e ψ_{20})
seções: set of <string>	$\delta_4^{[V3,3]}$: $[T_{[V3,3]} \bullet \text{secoes}] \equiv T_{\text{autores\_rel}} \bullet fk_1^{-1} \bullet fk_4^{-1} \bullet \text{nome}$(de $\psi_4^{[V1,4]}$, $\psi_{21}^{[V1,4]}$ e ψ_4)
matérias: set of <$T_{[materia]}$>	$\delta_5^{[V3,3]}$: $[T_{[V3,3]} \bullet \text{materias}] \equiv T_{\text{autores\_rel}} \bullet fk_1^{-1}$ (de $\psi_5^{[V1,4]}$ e ψ_{22})
titulo: string	$\delta_6^{[V3,3]}$: $[T_{[materia]} \bullet \text{titulo}] \equiv T_{\text{materias\_rel}} \bullet \text{titulo}$ (de $\psi_6^{[V1,4]}$ e ψ_9)
data_publicacao: string	$\delta_7^{[V3,3]}$: $[T_{[materia]} \bullet \text{data\_publicacao}] \equiv T_{\text{materias\_rel}} \bullet \text{data}$ (de $\psi_7^{[V1,4]}$ e ψ_{10})
resumo: string	$\delta_8^{[V3,3]}$: $[T_{[materia]} \bullet \text{resumo}] \equiv T_{\text{materias\_rel}} \bullet \text{resumo}$ (de $\psi_8^{[V1,4]}$ e ψ_{11})
Elos:	
Operações:	
Ordenação:	

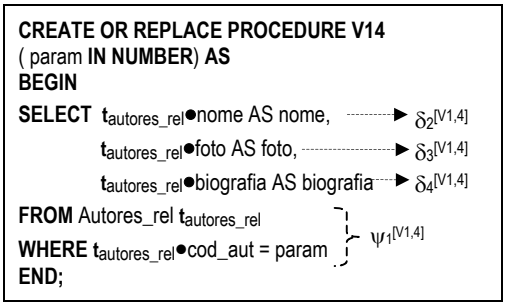
Passo 2: Gerar definições da VCN sobre esquema do banco de dados

- Definição SQL das VCN's do UID $\mathcal{U}_1$ sobre o esquema do banco de dados relacional

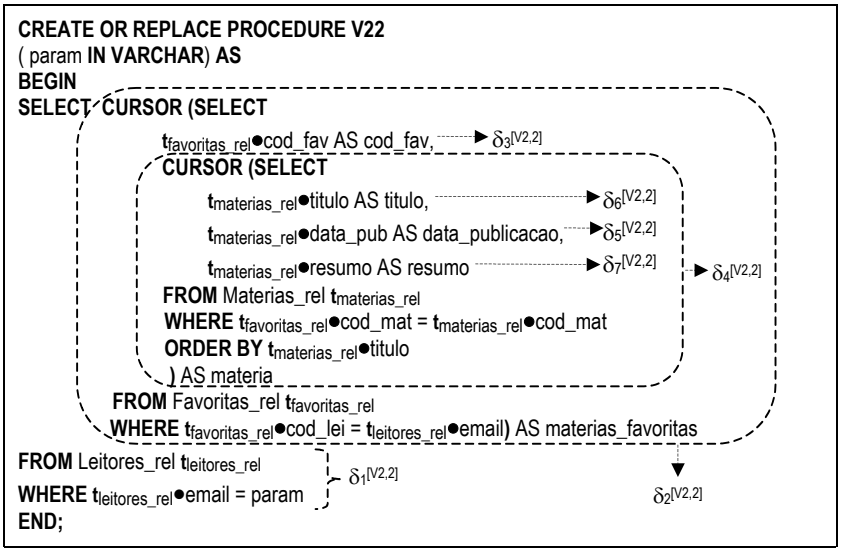
```
CREATE OR REPLACE PROCEDURE V11 AS
BEGIN
SELECT t_secoes_rel•cod_sec AS cod_sec,        ⇢ δ2[V1,1]
       t_secoes_rel•nome AS nome               ⇢ δ3[V1,1]
FROM Secoes_rel t_secoes_rel                   ⇢ δ1[V1,1]
ORDER BY t_secoes_rel•nome
END;
```

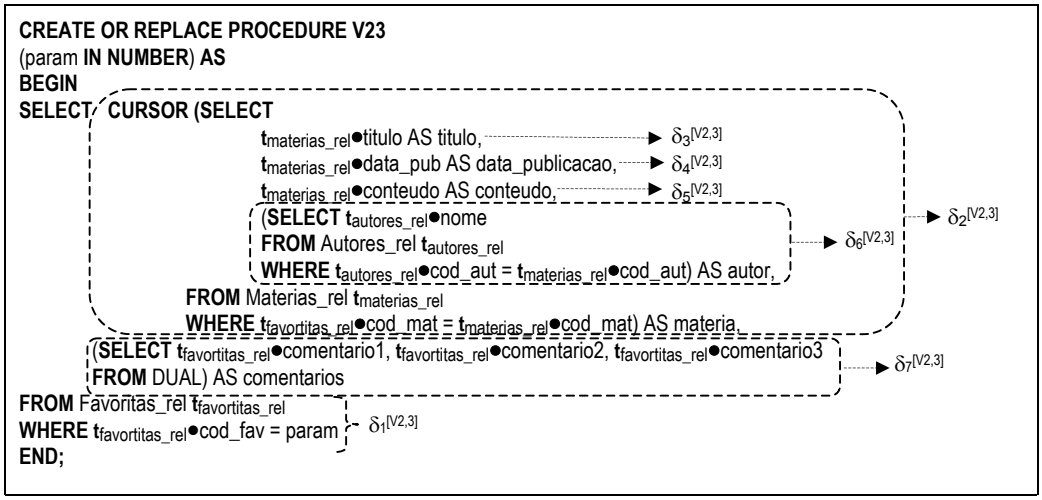
```
CREATE OR REPLACE PROCEDURE V12
( param IN NUMBER) AS
BEGIN
SELECT CURSOR (SELECT
         t_materias_rel•cod_mat AS cod_mat,        ⇢ δ3[V1,2]
         t_materias_rel•data_pub AS data_publiccao, ⇢ δ4[V1,2]
         t_materias_rel•titulo AS titulo,          ⇢ δ5[V1,2]   ⇢ δ2[V1,2]
         t_materias_rel•resumo AS resumo           ⇢ δ6[V1,2]
       FROM Materias_rel t_materias_rel
       WHERE t_materias_rel•cod_sec = t_secoes_rel•cod_sec
       ORDER BY t_materias_rel•titulo
       ) AS materias
FROM Secoes_rel t_secoes_rel                       ⇢ δ1[V1,2]
WHERE t_secoes_rel•cod_sec = param
END;
```



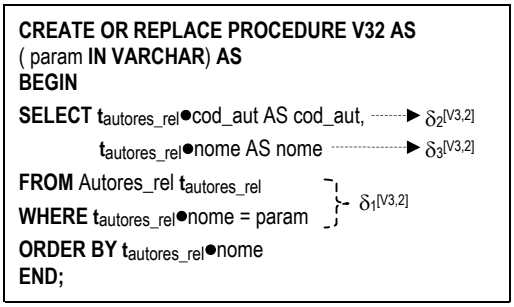


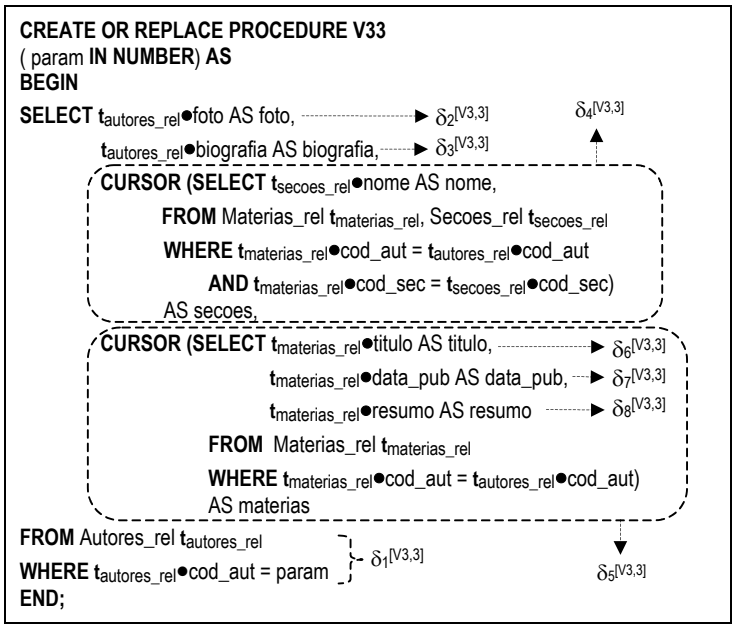
❖ Definição SQL das VCN's do UID $\mathcal{U}_2$ sobre o esquema do banco de dados relacional





❖ Definição SQL das VCN's do UID $\mathcal{U}_3$ sobre o esquema do banco de dados relacional





A.4.2. Implementação como Visões XML

Nesta abordagem, os dados das VCN's extraídos do banco de dados são representados no formato XML. Para cada VCN, é gerada uma página dinâmica (XSQL Page) que publica o conteúdo da VCN no formato XML. Como vimos, a implementação das VCN's pode ser feita em 3 camadas ou 2 camadas. Para exemplificar, geramos as páginas XSQL utilizando a implementação em 2 camadas, onde as consultas SQL são definidas sobre o esquema do banco de dados relacional. A seguir, apresentamos a XSQL *Page* gerada para cada VCN, além de um exemplo de documento XML transformado no formato da VCN.

❖ VCN's do UID $\mathcal{U}_1$

- VCN $\mathcal{V}_{1,1}$

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="UID1v11.xsl"?>
<xsql:query connection="xml" xmlns:xsql="urn:oracle-xsql">
SELECT t_secoes_rel●cod_sec as cod_sec,
       t_secoes_rel●nome as nome
FROM Secoes_rel t_secoes_rel
ORDER BY t_secoes_rel●nome
</xsql:query>
```

```
<?xml version="1.0" ?>
<Resultado>
  <Secao>
      <cod_sec>1</cod_sec>
      <nome>Engenharia de Software</nome>
  </Secao>
  <Secao>
      <cod_sec>2</cod_sec>
      <nome>Banco de Dados</nome>
  </Secao>
</Resultado>
```

- VCN $\mathcal{V}_{1,2}$

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="UID1v12.xsl"?>
<xsql:query connection="xml" xmlns:xsql="urn:oracle-xsql">
SELECT  CURSOR (SELECT t_materias_rel●cod_mat as cod_mat,
                       t_materias_rel●data_pub as data_publicacao,
                       t_materias_rel●titulo as titulo,
                       t_materias_rel●resumo as resumo
          FROM Materias_rel t_materias_rel
          WHERE t_materias_rel●cod_sec = t_secoes_rel●cod_sec
          ORDER BY t_materias_rel●titulo) AS materias
FROM Secoes_rel t_secoes_rel
WHERE t_secoes_rel●cod_sec = {@param}
</xsql:query>
```

```
<?xml version="1.0" ?>
<Resultado>
  <Materia>
      <data_ publicacao>01/10/1993</data_ publicacao>
      <titulo>Banco de Dados Relacionais</titulo>
      <resumo>Banco de Dados Relacionais...</resumo>
   </Materia>
  < Materia >
      <data_ publicacao>10/10/2002</data_ publicacao>
      <titulo>XML e Oracle</titulo>
      <resumo>XML e Oracle...</resumo>
   </Materia>
</Resultado>
```

- VCN $\mathcal{V}_{1,3}$

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="UID1v13.xsl"?>
<xsql:query connection="xml" xmlns:xsql="urn:oracle-xsql">
SELECT  t_materias_rel●titulo as titulo,
        t_materias_rel●data_pub as data_publicacao,
        t_materias_rel●conteudo as conteudo,
        CURSOR (SELECT t_autores_rel●cod_aut as cod_aut, t_autores_rel●nome as nome
                FROM Autores_rel t_autores_rel
                WHERE t_materias_rel●cod_aut = t_autores_rel●cod_aut) as autor,
        CURSOR (SELECT t_matrel_rel2●titulo as titulo, t_matrel_rel2●data_pub as data_publicacao
                FROM Materias_rel t_matrel_rel2, MatRel_rel t_matrel_rel
                WHERE t_matrel_rel●mat_rel = t_matrel_rel2●cod_mat
                      AND t_matrel_rel●cod_mat = t_materias_rel●cod_mat ) as mat_rel
FROM Materias_rel t_materias_rel
WHERE t_materias_rel●cod_mat = {@param}
</xsql:query>
```

```
<?xml version="1.0" ?>
<Resultado>
  <Materia>
      <titulo>XML e Oracle</titulo>
      <data_publicacao>10/10/2002</data_ publicacao>
      <conteudo>XML e Oracle ...</conteudo>
      <autor>
          <nome>A. Steven</nome>
      </autor>
      <mat_rel>
          <titulo>Banco de Dados XML Nativo</titulo>
          <data_publicacao>08/05/2003 </data_publicacao>
      </mat_rel>
      <mat_rel>
          <titulo>XSQL PAGES Publishing </titulo>
          <data_publicacao>01/03/2002 </data_publicacao>
      </mat_rel>
  </Materia>
</Resultado>
```

- VCN $\mathcal{V}_{1,4}$

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="UID1v14.xsl"?>
<xsql:query connection="xml" xmlns:xsql="urn:oracle-xsql">
SELECT t_autores_rel.nome as nome,
       t_autores_rel.foto as foto,
       t_autores_rel.biografia as biografia
FROM Autores_rel t_autores_rel
WHERE t_autores_rel.cod_aut = {@param}
</xsql:query>
```

```
<?xml version="1.0" ?>
<Resultado>
    <Autor>
        <nome>A. Steven</nome>
        <foto>asteven.jpg</foto>
        <biografia>...</biografia>
    </Autor>
</Resultado>
```

❖ VCN's do UID $\mathcal{U}_2$

- VCN $\mathcal{V}_{2,2}$

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="UID2v21.xsl"?>
<xsql:query connection="xml" xmlns:xsql="urn:oracle-xsql">
SELECT  CURSOR (SELECT
                    t_favoritas_rel●cod_fav AS cod_fav,
                    CURSOR (SELECT
                        t_materias_rel●titulo AS titulo,
                        t_materias_rel●data_pub AS data_publicacao,
                        t_materias_rel●resumo AS resumo
                    FROM Materias_rel t_materias_rel
                    WHERE t_favoritas_rel●cod_mat = t_materias_rel●cod_mat
                    ORDER BY t_materias_rel●titulo
                    ) AS materia
               FROM Favoritas_rel t_favoritas_rel
               WHERE t_favoritas_rel●cod_lei = t_leitores_rel●email) AS materias_favoritas
FROM Leitores_rel t_leitores_rel
WHERE t_leitores_rel●email = {@param}
</xsql:query>
```

```
<?xml version="1.0" ?>
<Resultado>
  <MateriaFavorita>
      <titulo>XML e Oracle</titulo>
      <data_publicacao>10/10/2002</data_publicacao>
      <resumo>XML e Oracle ...</resumo>
</ MateriaFavorita>
</Resultado>
```

- VCN $\mathcal{V}_{2,3}$

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="UID2v22.xsl"?>
<xsql:query connection="xml" xmlns:xsql="urn:oracle-xsql">
SELECT CURSOR(SELECT t_materias_rel●titulo as título,
                    t_materias_rel●data_pub as data_publicacao,
                    t_materias_rel●conteudo as conteudo,
                    (SELECT t_autores_rel●nome
                    FROM Autores_rel t_autores_rel
                    WHERE t_autores_rel●cod_aut = t_materias_rel●cod_aut) as autor)
               FROM Materias_rel t_materias_rel
               WHERE t_favoritas_rel●cod_mat = t_materias_rel●cod_mat) AS materia,
        CURSOR(SELECT t_favoritas_rel●comentario1, t_favoritas_rel●comentario2, t_favoritas_rel●comentario3
        FROM DUAL) AS comentarios
FROM Favoritas_rel t_favoritas_rel
WHERE t_favoritas_rel●cod_fav = {@param}
</xsql:query>
```

```
<?xml version="1.0" ?>
<Resultado>
  <MateriaFavorita>
      <titulo>XML e Oracle</titulo>
      <data_de_publicacao>10/10/2002</data_ publicacao>
      <conteudo>XML e Oracle ...</conteudo>
      <autor>
            <nome>A. Steven</nome>
       </autor>
      <comentarios>
            <comentario1>... </comentario1>
             <comentario2>... </comentario2>
             <comentario3>... </comentario3>
       </comentarios>
   </ MateriaFavorita>
</Resultado>
```

❖ VCN's do UID $\mathcal{U}_3$

- VCN $\mathcal{V}_{3,2}$

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="UID2v31.xsl"?>
<xsql:query connection="xml" xmlns:xsql="urn:oracle-xsql">
SELECT t_autores_rel●cod_aut as cod_aut,
       t_autores_rel●nome as nome,
       t_autores_rel●foto as foto,
       t_autores_rel●biografia as biografia
FROM Autores_rel t_autores_rel
WHERE t_autores_rel●nome = {@param}
ORDER BY t_autores_rel●nome
</xsql:query>
```

```
<?xml version="1.0" ?>
<Resultado>
  <Autor>
      <nome>A. Steven</nome>
      <foto>asteven.jpg</foto>
      <biografia>...</biografia>
   </Autor>
  <Autor>
      <nome> J. K.  Steven</nome>
      <foto>jksteven.jpg</foto>
      <biografia>...</biografia>
   </Autor>
</Resultado>
```

- VCN $\mathcal{V}_{3,3}$

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="UID2v32.xsl"?>
<xsql:query connection="xml" xmlns:xsql="urn:oracle-xsql">
SELECT t_autores_rel●foto as foto,
       t_autores_rel●biografia as biografia,
       t_autores_rel●conteudo as conteudo,
       CURSOR (SELECT t_secoes_rel●nome as nome
               FROM Materias_rel t_materias_rel, Secoes_rel t_secoes_rel
               WHERE t_materias_rel●cod_aut = t_autores_rel●cod_aut
                     AND t_materias_rel●cod_sec = t_secoes_rel.cod_sec) as secoes,
       CURSOR (SELECT t_materias_rel●titulo as titulo,
                      t_materias_rel●data_pub as data_publicacao,
                      t_materias_rel●resumo as resumo
               FROM  Materias_rel t_materias_rel
               WHERE t_materias_rel●cod_aut = t_autores_rel●cod_aut) as materias
FROM Autores_rel t_autores_rel
WHERE t_autores_rel●cod_aut = {@param}
</xsql:query>
```

```
<?xml version="1.0" ?>
<Resultado>
  <Autor>
      <foto>asteven.jpg</foto>
      <biografia>...</biografia>
      <Secao>
          <nome>Banco de Dados</nome>
          <nome>XML</nome>
      </Secao>
      <Materia>
          <titulo>XML e Oracle</titulo>
          <data_publicacao>10/10/2002</data_ publicacao>
          <resumo>XML e Oracle ...</resumo>
      <Materia>
      <Materia>
          <titulo>Banco de Dados XML Nativo</titulo>
          <data_publicacao>25/04/2003</data_ publicacao>
          <resumo> Banco de Dados XML Nativo ....</resumo>
      <Materia>
  </Autor>
</Resultado>
```