Universidade Federal do Ceará
Centro de Ciências
Departamento de Computação
Mestrado e Doutorado em Ciência da Computação

Um Framework para Sistemas de Mediação Baseados em XML

Dissertação de Mestrado

George Marcel Lima Teixeira

Fortaleza - Ceará
Outubro/2009

Universidade Federal do Ceará
Centro de Ciências
Departamento de Computação
Mestrado e Doutorado em Ciência da Computação

Um Framework para Sistemas de Mediação Baseados em XML

Dissertação apresentada à Coordenação do Programa de Mestrado e Doutorado em Ciência da Computação da Universidade Federal do Ceará, como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Orientadora:

Profa. Dra. Vânia Maria Ponte Vidal

George Marcel Lima Teixeira

Universidade Federal do Ceará
Centro de Ciências
Departamento de Computação
Mestrado e Doutorado em Ciência da Computação

Um Framework para Sistemas de Mediação Baseados em XML

Dissertação de Mestrado

George Marcel Lima Teixeira

Aprovada em __/__/_____

Banca Examinadora

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

Profa. Dra. Marta Lima de Queirós Mattoso
Universidade Federal do Rio de Janeiro – UFRJ

Prof. Dr. José Antônio Fernandes de Macêdo
Universidade Federal do Ceará – UFC

A minha família.

Agradecimentos

A Deus, que me concedeu a vida e esta oportunidade.

Aos meus pais Heitor e Luiza e aos meus irmãos Renan e Ciro, que em todos os momentos acreditaram e me deram forças para concluir este trabalho.

A minha esposa Virgínia, que com muito amor, carinho e companheirismo sempre me incentivou a seguir em frente.

A minha orientadora Professora Vânia Vidal, pela dedicação, paciência, ensinamentos e lições para a vida.

Aos membros da banca examinadora, Professora Marta Mattoso e Professor José Antônio Macêdo, por terem aceitado o convite e por contribuírem para a melhoria deste trabalho.

Aos Professores Javam e Berna pelo apoio para a conclusão desta caminhada.

Aos amigos João e Fernando, por tudo o que me ajudaram, especialmente pela paciência em retransmitir à distância importantes sugestões para esta dissertação.

Aos amigos do grupo Arida, Eveline, Luiz, Fábio e Valdiana que de várias formas também colaboraram com este trabalho.

Aos funcionários do Departamento de Computação, especialmente ao Orley, que sempre esteve pronto para atender nossas demandas acadêmicas.

A todos os meus familiares e amigos que torceram pela conquista deste objetivo.

A Universidade Federal do Ceará (UFC) e ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) por tornarem possível a realização deste mestrado.

Resumo

Um sistema de integração de dados baseado em mediação provê para o usuário uma interface uniforme (visão de mediação) de diferentes fontes de dados via um modelo comum, permitindo que fontes de dados autônomas, heterogêneas e distribuídas possam ser consultadas através desta visão, sem a necessidade do usuário conhecer os procedimentos de acesso, extração e integração dos dados de cada fonte.

A especificação de visões de mediação – VMs (incluindo o formalismo para definir o mapeamento entre o esquema da visão e os esquemas das fontes) e o processamento de consultas sobre estas VMs são problemas fundamentais que devem ser tratados pelos sistemas de mediação. De maneira geral, os formalismos encontrados na literatura para a definição de mapeamentos sobre dados com estruturas aninhadas (como o XML) são complexos, pois precisam ser flexíveis o suficiente para tratar a heterogeneidade das fontes de dados e expressar corretamente a semântica desejada na definição da VM. Por conseqüência, os algoritmos propostos para o processamento de consultas são também complexos, pois precisam utilizar, em tempo de execução, estes mapeamentos, definidos diretamente sobre a estrutura de fontes de dados heterogêneas.

Neste trabalho, propomos um *framework* para sistemas de mediação que utiliza XML para a codificação e transporte de dados, e serviços web como interfaces para realizar a comunicação entre os clientes e as fontes de dados. O *framework* adota uma arquitetura de três níveis de esquemas, onde uma VM é definida em termos de visões XML (exportadas pelas fontes de dados via serviços web), as quais consistem de fragmentos (verticais, horizontais ou híbridos) da VM. A arquitetura proposta objetiva simplificar a integração de dados de fontes heterogêneas e distribuídas, trazendo o problema do processamento de consultas para um contexto de integração de fontes de dados homogêneas, uma vez que as visões exportadas pelas fontes são fragmentos da VM, gerados automaticamente a partir da sua especificação.

As principais contribuições deste trabalho são: (i) Um formalismo para a especificação de VMs diretamente sobre fontes de dados heterogêneas e distribuídas (incluindo um formalismo simples, mas expressivo, para a definição de mapeamentos); (ii) Um processo para, a partir da especificação, gerar automaticamente VMs definidas em termos das visões exportadas pelas fontes (arquitetura de três níveis de esquemas); e (iii) uma metodologia para o processamento de consultas sobre estas VMs.

Sumário

Lista de Figuras

Lista de Acrônimos

A

AC – Assertiva de Correspondência

ACC – Assertiva de Correspondência de Caminho

ACG – Assertiva de Correspondência Global ou de Coleção

P

PC – Processo-Consulta

PF – Processo-Filtro

$P_I(Q_M)$ – Partição de Integração da consulta Q_M

PJM – Processo-Junção Múltipla

PO – Processo-Outer Union

PU – Processo-Union

V

VLE – Visão Local Exportada

VM – Visão de Mediação

W

$W_F(Q_M)$ – Workflow Final de execução da consulta Q_M

$W_I(Q_M)$ – Workflow Inicial de execução da consulta Q_M

Capítulo I

Introdução

1.1. Motivação

O advento da Internet impulsionou a publicação e a troca de informações entre aplicações e trouxe consigo a necessidade de mecanismos eficientes para a integração de dados. Empresas e entidades governamentais precisam cada vez mais integrar informações que, mesmo relativas a domínios comuns, estão armazenadas em fontes heterogêneas e distribuídas, criadas de forma independente.

O principal objetivo de uma solução de integração é permitir que usuários consultem simultaneamente múltiplas fontes de dados heterogêneas e distribuídas, através de uma única interface de consultas, mantendo transparentes os procedimentos de acesso, extração e integração dos dados. Pesquisadores têm estudado intensamente sobre arquiteturas e ferramentas para a integração de dados [7][13][25][35][51] e proposto sistemas de integração [5][6][21][27][31][33].

De acordo com [30], um sistema de integração de dados (SID) pode ser descrito em termos dos seus principais componentes por uma tripla <G, S, M>, onde G representa um esquema global, S representa os esquemas das fontes de dados e M representa os mapeamentos entre G e S, ou seja, as correspondências entre os elementos de G e S. É através do esquema global que os usuários podem realizar consultas às fontes de dados de maneira uniforme, centralizada e transparente.

Duas abordagens clássicas [2] são, normalmente, seguidas pelos sistemas de integração: (i) a abordagem *materializada*, onde os dados do esquema global são extraídos das fontes e armazenados em repositórios locais, sobre os quais as consultas são realizadas; ou (ii) a abordagem *virtual*, onde os dados são recuperados diretamente das fontes quando o sistema precisa responder a uma consulta sobre o esquema global.

Uma arquitetura convencional de sistemas de integração que segue a abordagem virtual é aquela baseada em mediadores [48]. Nessa arquitetura, o esquema global pode também ser chamado de *Esquema de Mediação*. Os mediadores são os responsáveis por disponibilizar o esquema de mediação, reescrever consultas aplicadas sobre esse esquema em subconsultas sobre as fontes de dados e integrar os resultados obtidos.

Seja qual for a abordagem adotada, é fundamental para os sistemas de integração que se estabeleça como os elementos do esquema global estão relacionados com os elementos dos esquemas locais. Portanto, são definidos os mapeamentos (M) entre o esquema global (G) e os esquemas locais (S), os quais possibilitam tanto materializar o esquema global (abordagem materializada) quanto reescrever consultas sobre o esquema global em subconsultas sobre as fontes locais (abordagem virtual). Os enfoques de definição de mapeamentos mais comumente encontrados na literatura [30] são: (i) *Global-As-View*, onde o esquema global é definido pelos mapeamentos como uma visão sobre os esquemas locais, e (ii) *Local-As-View*, onde os mapeamentos definem os esquemas das fontes locais como visões sobre o esquema global.

Com o incremento constante do número de fontes que disponibilizam dados na Web, os sistemas de integração tradicionais precisaram sofrer adaptações para acessar, trocar e integrar dados de forma eficiente dentro deste novo paradigma. A Linguagem XML [47] tem surgido como formato universal para publicação e a troca de dados na Web, e tem a XQuery [47] como principal proposta de linguagem para consulta sobre documentos XML. Os serviços web (*web services*) têm quebrado significativas barreiras para a interoperabilidade de aplicações, facilitando o acesso, garantindo segurança e possibilitando a auto-descrição das capacidades ou funcionalidades suportadas.

Neste contexto, alguns trabalhos [6][15][22][27][31][50] passaram a propor soluções baseadas em XML, utilizando XQuery como linguagem de consulta sobre seus esquemas globais. De maneira geral, os formalismos encontrados na literatura para a definição de mapeamentos sobre dados com estruturas aninhadas (como o XML) são complexos [20], pois precisam ser flexíveis o suficiente para tratar a heterogeneidade das fontes de dados e expressar corretamente a semântica desejada na definição do esquema global. Por conseqüência, os algoritmos propostos para o processamento de consultas são também complexos [49][50], pois precisam utilizar, em tempo de execução, mapeamentos de difícil manipulação, definidos diretamente sobre estruturas aninhadas. Muitas vezes, estes algoritmos também ignoram estratégias de otimização.

1.2. Objetivo e Contribuições

O objetivo desta dissertação é contribuir com uma solução para a integração de dados de fontes heterogêneas e distribuídas, trazendo o problema do processamento de consultas para um contexto de integração de fontes de dados homogêneas, onde algumas vantagens podem ser obtidas, conforme será visto a seguir.

Para isso, propomos um *framework* para sistemas de integração de dados baseados em mediação e que utilizam visões XML [37][42] como esquemas globais. O *framework* adota uma arquitetura de três níveis de esquemas, conforme exibido na Figura 1.1(b).

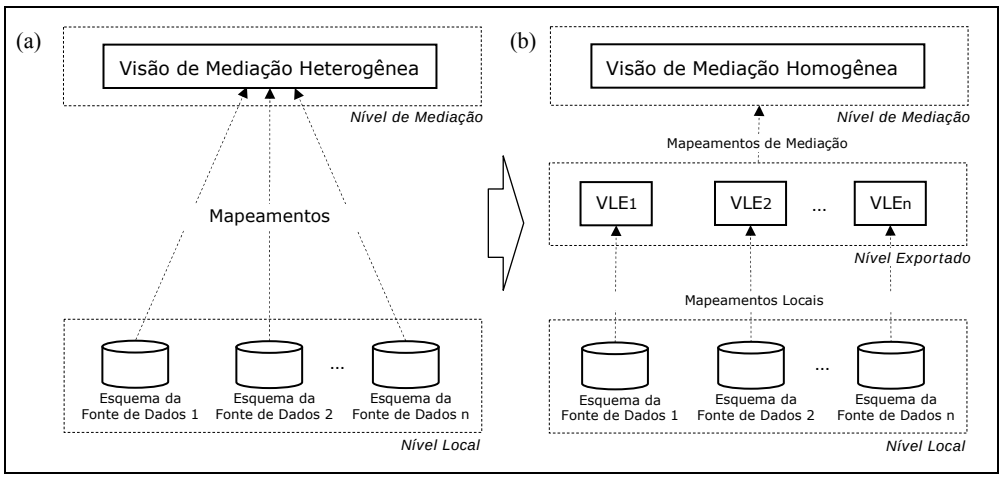


Figura 1.1 – (a) VM Heterogênea; (b) VM Homogênea (arquitetura de três níveis de esquemas)

O ***Nível Local*** contém os esquemas de fontes de dados heterogêneas e distribuídas, onde os dados estão realmente armazenados. O ***Nível Exportado*** contém um conjunto de visões XML chamadas de *Visões Locais Exportadas* (VLEs), onde cada VLE é definida em termos do esquema de uma única fonte de dados local. O ***Nível de Mediação*** contém um conjunto de visões XML chamadas de *Visões de Mediação* (VMs), onde cada VM é definida em termos de um conjunto de VLEs. As VLEs são fragmentos homogêneos (horizontais, verticais ou híbridos)[32] das VMs e, por isso, estas VMs são também chamadas de *Visões de Mediação Homogêneas*. Uma VM homogênea é gerada a partir de uma visão XML equivalente definida diretamente em termos das fontes de dados distribuídas e heterogêneas, chamada de *Visão de Mediação Heterogênea*, conforme mostrado na Figura 1.1(a).

A arquitetura de três níveis de esquemas permite quebrar o problema do processamento de consultas em dois subproblemas: (i) o processamento de consultas no contexto de fontes de dados homogêneas, ou seja, "Como transformar uma consulta sobre a VM em subconsultas sobre as VLEs (as quais são fragmentos homogêneos da VM) e integrar os resultados obtidos"; e (ii) o processamento de consultas no contexto da publicação de dados de cada fonte através de uma visão XML local, ou seja, "Como transformar subconsultas sobre as VLEs em consultas sobre os seus esquemas locais".

O primeiro problema é tratado pelo mediador do *framework*, que publica a VM, e o segundo problema é tratado por tradutores locais (*wrappers*), que publicam as VLEs. Assim, os principais passos do processamento de uma consulta sobre uma VM homogênea pode ser resumido conforme se segue: (i) A consulta é decomposta pelo mediador em subconsultas sobre as VLEs; (ii) As subconsultas sobre as VLEs são traduzidas pelos tradutores locais em consultas sobre os esquemas das fontes de dados; e (iii) os resultados das subconsultas são retornados ao mediador, onde é realizada a composição do resultado final.

O fato das VLEs geradas representarem fragmentos homogêneos da VM favorece o algoritmo de processamento de consultas, permitindo: (i) realizar uma decomposição simples de consultas sobre as VMs em subconsultas sobre as VLEs, sem a necessidade de verificar mapeamentos de esquemas em tempo de execução; (ii) realizar a integração de resultados de subconsultas apenas com operações de união e junção; e (iii) adotar estratégias de otimização, como a geração de planos de execução de consultas com base na relevância e distribuição dos dados nas fontes, na redução do volume de dados trafegados e no custo computacional das operações de integração.

O *framework* proposto é, portanto, composto por elementos que possibilitam a geração de VMs homogêneas e o processamento de consultas sobre estas visões, a saber: (i) um formalismo e um processo para a definição de uma VM heterogênea (incluindo um formalismo simples, mas expressivo, para a definição de mapeamentos, seguindo o enfoque *Global-As-View*); (ii) um algoritmo que transforma a definição de uma VM heterogênea em uma VM homogênea equivalente, definida em termos das VLEs. Este mesmo algoritmo gera automaticamente as VLEs a partir da definição da VM heterogênea; e (iii) um algoritmo que implementa uma metodologia para o processamento de consultas sobre VMs homogêneas.

Em síntese, a geração de VMs homogêneas (de acordo com a arquitetura de três níveis de esquemas, com a geração automática das VLEs) e a metodologia para o processamento de consultas sobre as VMs homogêneas constituem as contribuições chave deste trabalho.

No *framework* proposto, o mediador consiste de um serviço web que disponibiliza métodos para consulta às VMs e os tradutores consistem de serviços web que disponibilizam métodos para consultas às VLEs. Desta forma, o plano de execução de uma consulta consiste de um *workflow* de processos, controlado pelo mediador, que disparam a execução de serviços web.

É importante observar que a proposta deste trabalho está focada no problema de integração de dados de fontes que são previamente conhecidas por um projetista. Não há descoberta automática de informações ou mapeamentos entre esquemas. Nossa proposta é aplicável, por exemplo, quando se deseja integrar informações de entidades governamentais que têm dados de um mesmo domínio de aplicação distribuídos por vários municípios ou estados; ou uma instituição com seus dados operacionais distribuídos em filiais. O projetista é responsável por especificar as VMs e identificar/definir restrições entre as fontes de dados que permitam otimizar o processamento de consultas e/ou definir VMs com maior riqueza de informações.

Duas características importante relacionadas à integração de dados de fontes heterogêneas e distribuídas podem ainda ser destacadas neste trabalho:

- ✓ O formalismo utilizado para a especificação de VMs suporta a definição de *ligações* entre os esquemas das fontes de dados, permitindo ao projetista definir VMs com uma maior riqueza de informações (informações de uma fonte de dados podem ser complementadas por informações de fontes referenciadas por ligações). Estas ligações dão suporte à implementação das operações de junção tratadas neste trabalho.
- ✓ Todas as operações de junção seguem uma estratégia baseada na operação de semi-junção [17], objetivando, quando for o caso, reduzir o volume de dados trafegados das fontes para o mediador e, por conseqüência, melhorar o desempenho do processamento de consultas.

1.3. Organização da Dissertação

Os capítulos a seguir estão organizados da seguinte forma. No Capítulo II, são discutidas as principais arquiteturas de sistemas de integração de dados e apresentados alguns trabalhos relacionados com esta dissertação. No Capítulo III, o *framework* proposto é apresentado em maiores detalhes. No Capítulo IV, são fornecidos os fundamentos teóricos necessários à apresentação da abordagem para especificação e geração de VMs e da metodologia para o processamento de consultas. No Capítulo V, é apresentada a abordagem para especificação e geração de VMs. No Capítulo VI, é apresentada a metodologia para o processamento de consultas sobre VMs. No Capítulo VII, são apresentadas as conclusões e sugestões para trabalhos futuros.

Capítulo II

Integração de Dados

Neste capítulo, são discutidos aspectos relevantes do problema de integração de dados. São apresentadas abordagens clássicas de integração (virtual e materializada), arquiteturas de sistemas de integração de dados e os trabalhos relacionados com esta dissertação.

2.1. Abordagens para Integração de Dados

Uma solução de integração dados deve permitir aos seus usuários consultar múltiplas fontes de dados através de uma interface centralizada e uniforme, sem a necessidade de conhecer os procedimentos de acesso, extração e integração dos dados de cada fonte. Surgem, então, os sistemas de integração de dados, os quais disponibilizam um esquema integrado das fontes de dados, chamado de esquema global, através do qual as fontes poderão ser consultadas. Os sistemas de integração devem tratar, de forma transparente para o usuário, problemas de heterogeneidade (estrutural, conceitual e tecnológica), distribuição e autonomia das fontes durante a execução de consultas sobre o esquema global.

Os sistemas de integração de dados seguem, normalmente, duas abordagens clássicas [2]: virtual ou materializada. A abordagem materializada extrai os dados de fontes distintas e materializa-os localmente em repositórios chamados *datawarehouses*. As consultas são realizadas sobre a base materializada e, desta forma, apresentam melhor desempenho em relação à abordagem virtual. A desvantagem dessa abordagem é a necessidade de se manter a base materializada sempre atualizada. Em alguns casos, é necessária a presença de um administrador para a atualização. Caso contrário, poderão ser geradas respostas com informações desatualizadas.

Na abordagem virtual, os dados são recuperados diretamente das fontes quando o sistema de integração precisa responder a uma consulta, ou seja, sistemas que seguem esta abordagem enviam as consultas diretamente às fontes e, após os resultados individuais obtidos, integram os dados e fornecem o resultado final ao usuário ou à aplicação que os requisitou. Os dados acessados por esta abordagem estão sempre

atualizados, entretanto, sabe-se que os custos de processamento das consultas e de acesso às fontes são fatores consideráveis na sua escolha.

A escolha da abordagem a ser seguida está diretamente relacionada à arquitetura e às características da aplicação de integração. Em alguns casos, as duas abordagens podem até ser utilizadas concomitantemente dentro de uma mesma solução.

2.2. Taxonomia das Arquiteturas para Integração de Dados

Dada a importância de conceitos como heterogeneidade, distribuição e autonomia no estudo e classificação das arquiteturas de integração de dados, será apresentada, a seguir, uma breve descrição destes conceitos.

- Heterogeneidade: pode ocorrer em vários níveis dos sistemas de integração de dados como, por exemplo, heterogeneidades de hardware, de protocolos de rede, de modelo de dados e linguagens de consulta aos bancos de dados, entre outras. Os próprios dados podem apresentar heterogeneidade estrutural, relacionada aos diferentes esquemas ou estruturas de armazenamento desses dados em cada fonte, e heterogeneidade semântica, relacionada ao fato do mesmo conceito do mundo real poder ser representado de variadas formas em fontes de dados distintas. Os sistemas de integração de dados devem resolver problemas de heterogeneidade.

- Distribuição: diz respeito à alocação física dos dados em diversas fontes de informação geograficamente distribuídas. A distribuição de dados demanda um meio de comunicação através do qual o sistema de integração pode trocar informações com as fontes de dados.

- Autonomia: refere-se ao nível de independência de operação de cada fonte de dados que participe de um sistema de integração. Pode ser classificada em: (i) autonomia de projeto: as fontes podem alterar seus modelos de dados ou o esquema dos seus dados a qualquer instante; (ii) autonomia de controle de execução: as fontes, quando for o caso, definem a ordem de execução de suas transações; (iii) autonomia de comunicação: garante a liberdade de quando e como as fontes deverão responder às solicitações; e (iv) autonomia de associação: as fontes decidem quando e como suas operações e dados serão compartilhados com outras aplicações e arquiteturas de integração de dados.

Nas próximas seções, algumas arquiteturas clássicas serão brevemente discutidas.

2.2.1. Bancos de Dados Federados

Caracterizados por um conjunto de SGBD's autônomos que cooperam para o compartilhamento parcial e controlado dos seus dados [10]. Utilizam um modelo de dados comum e tratam dados estruturados. O Sistema Gerenciador de Bancos de Dados Federados (SGBDF) é o responsável pelo gerenciamento de cada SGBD participante da federação. O esquema federado representa a integração dos esquemas dos bancos participantes e pode ser utilizado pelos usuários para definir sobre este um novo esquema, denominado esquema externo, que atenda aos requisitos de suas aplicações. Bancos de dados federados possibilitam o acesso transparente a dados distribuídos e heterogêneos, mas apresentam como desvantagem a dificuldade para geração e manutenção do esquema federado.

2.2.2. Bancos de Dados Múltiplos

Considerada a primeira abordagem para o compartilhamento entre SGBD's heterogêneos. Não utiliza o conceito de esquema integrado que represente a integração dos esquemas dos bancos locais, mas baseia-se na utilização de linguagens de bancos de dados múltiplos que possibilitam ao usuário identificar como e de quais bancos de dados as informações serão obtidas [25]. Desta forma, apresenta como desvantagens: (i) a necessidade do usuário conhecer a localização exata dos bancos de dados; (ii) compreender detalhes dos esquemas destes bancos e (iii) detectar e resolver conflitos causados por heterogeneidades de diversas naturezas.

2.2.3. Arquitetura de Mediadores

Em sistemas baseados em mediação [48], as consultas são realizadas sobre um esquema integrado das fontes de dados, chamado neste trabalho de *Esquema de Mediação*. Ao receber uma consulta sobre este esquema, os mediadores realizam a sua reescrita em subconsultas que serão distribuídas pelas diversas fontes de dados que estão sendo integradas. Como as fontes podem estar baseadas em tecnologias, formatos e modelos completamente distintos, os mediadores normalmente se utilizam de tradutores (*wrappers*) para o acesso e execução destas consultas sobre os dados locais. Após obter os resultados locais, os mediadores consolidam o resultado integrado e retornam ao usuário a resposta da consulta realizada sobre o esquema de mediação.

A camada de mediação do sistema pode conter vários mediadores, que acessam e consultam diretamente as fontes de dados (através dos tradutores) ou até mesmo consultam outros mediadores, formando uma arquitetura multicamadas.

Na figura 2.1 é exibida uma arquitetura com um único mediador, a qual também será utilizada neste trabalho. De forma resumida, os principais passos do processamento de consultas nesta arquitetura são os seguintes:

Passo 1 – O usuário realiza uma consulta sobre o esquema de mediação publicado pelo sistema, mais especificamente, publicado pelo mediador.

Passo 2 – O mediador realiza a reescrita da consulta principal em subconsultas que serão enviadas aos tradutores das fontes locais. Os tradutores executam as subconsultas sobre as fontes de dados e retornam os resultados para o mediador.

Passo 3 – O mediador realizada a integração dos resultados das subconsultas e apresenta o resultado final ao usuário.

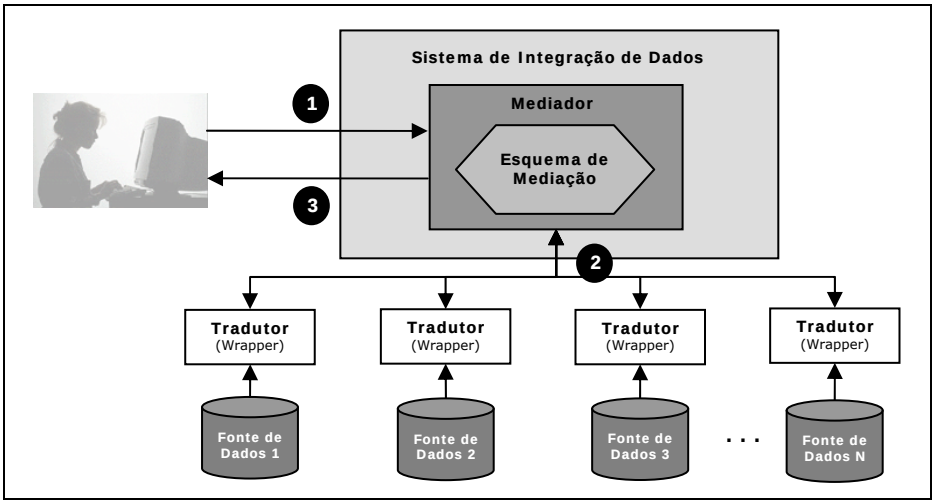


Figura 2.1 – Arquitetura de um Sistema de Integração baseado em Mediação

2.3. Trabalhos Relacionados

O problema da integração de dados tem sido amplamente estudado por mais de três décadas [30][51]. Diversas propostas para prover soluções de integração foram apresentadas e algumas resultaram na implementação de sistemas de integração de dados. Independentemente da abordagem, arquitetura e tecnologia adotada, estes sistemas têm desafios comuns a serem tratados, como: (i) a linguagem de consulta; (ii) o formalismo para a definição de mapeamentos; (iii) a eficiência dos algoritmos de processamento de consultas. A seguir, apresentamos alguns trabalhos relacionados com esta dissertação.

O TSIMMIS (*The Stanford-IBM Manager of Multiple Information Sources*)[21] é um projeto desenvolvido pela Universidade de Stanford em parceria com o IBM Almaden Research Center. Seu objetivo é fornecer ferramentas para acessar, de forma integrada, múltiplas fontes de informação e garantir que a informação obtida seja consistente. Utiliza uma arquitetura de mediadores/tradutores. Os tradutores exportam, a partir dos dados das fontes, objetos no modelo de dados OEM (*Object Exchange Model*). Uma importante contribuição do projeto é a implementação de ferramentas que geram automaticamente tradutores e mediadores. Entretanto, tarefas normalmente desempenhadas pelo mediador, como a compensação de capacidades de consulta, são realizadas pelos tradutores, deixando o passo de tradução mais pesado. Além disso, sua linguagem de consulta é bastante complexa requerendo uma ferramenta específica para facilitar a tradução de fragmentos da consulta principal nos tradutores.

O MIX (*Mediation of Information using XML*) [6] utiliza XML como modelo de dados comum e DTD's para a representação dos esquemas. Seus tradutores exportam dados em XML para o mediador, que realiza a integração. Propõe uma linguagem declarativa de alto nível chamada XMAS (XML Matching and Structuring Language) para definir as consultas e implementa uma interface gráfica chamada BBQ (Blended Browsing and Querying) através da qual o usuário navega pelos esquemas das fontes e pode montar sua consulta. O funcionamento do MIX exige apenas que os dados das fontes sejam representados por uma estrutura lógica em XML, não precisando ser convertidos do seu modelo original para XML em tempo de projeto. Entretanto, para montar uma consulta utilizando a interface gráfica, o usuário é responsável por definir as junções entre as relações de bases distintas, não ficando transparente a forma de integração dos dados.

O DENODO [33] é um sistema de integração baseado em mediadores que permite uma rápida e poderosa extração e combinação de informações de várias fontes heterogêneas, sejam estruturadas ou semi-estruturadas. Os tradutores que acessam as fontes são gerados de forma semi-automática e exportam, a partir das fontes, uma combinação de relações, chamadas relações base, seguindo um modelo semelhante ao relacional, onde registros e coleções são suportados. Cada tradutor deve prover relações base de forma que, quando consultadas pelo mediador, comportem-se como tabelas em um banco relacional. Por possuir tradutores que exportam dados num modelo semelhante ao relacional (*relacional-like*), o sistema perde características de enriquecimento semântico dos dados em relação a modelos como o XML, adotado pela

maioria dos sistemas mais recentes. Com XML, é possível definir tipos complexos e estruturados em vários níveis de forma a aprimorar a qualidade da resposta e facilitar os procedimentos de integração dos dados. Além disso, o acesso a fontes de informação na Web é feito por uma linguagem própria chamada NSEQL.

O ARTEMIS [5] é um ambiente para integração de dados baseado no uso de ontologias. A utilização de ontologias para a integração de dados vêm sendo tratada em diversas propostas [8][11][14], dada a sua maior expressividade para representar um domínio de conhecimento e considerar, além das características estruturais, os relacionamentos semânticos do domínio e a possibilidade de inferir novos conhecimentos [28].

A abordagem seguida pelo ARTEMIS é virtual e permite a integração semântica de fontes heterogêneas, sejam estruturadas ou semi-estruturada. Seu mecanismo é baseado na construção de uma representação semanticamente rica das fontes integradas. Para representar o conhecimento de integração (descrição do processo de integração e os resultados que são produzidos) é gerada uma ontologia de domínio (*domain ontology*). Nesta ontologia, o conhecimento de integração está organizado em conformidade com um esquema de mapeamento semântico (*semantic mapping scheme*), que guarda informações de conceitos das fontes que estão semanticamente relacionados (*clusters*); e um esquema de mediação (*mediation scheme*), que guarda os conceitos do esquema global obtidos das fontes locais e mantém as regras de mapeamento (*mapping rules*) entre a estrutura dos conceitos globais e das fontes de dados.

Em síntese, o ARTEMIS trabalha num processo de integração *bottom-up* , ou seja, a partir dos esquemas das fontes de dados gera um esquema global para representar a integração de todos os conceitos (relações) existentes nas fontes. Alguns outros trabalhos [8] apresentam soluções semelhantes ao ARTEMIS, seguindo um processo *bottom-up* para gerar uma representação integrada das informações presentes nas fontes. Em [14], uma ontologia pré-existente é utilizada como entrada e serve de referência na descoberta de conceitos semelhantes entre as fontes e na construção da visão global.

Em [49] foram apresentadas as seguintes contribuições para a integração de dados baseada em XML: (i) formalismo para a definição de mapeamentos complexos entre uma visão de integração (*target schema*) e as fontes de dados (*source schemas*); (ii) formalismo para a definição de restrições sobre o *target schema*; (iii) algoritmos para a reescrita de consultas e a combinação dos resultados obtidos das fontes, de acordo com as restrições sobre o *target schema*, gerando uma resposta final semanticamente correta.

O formalismo de mapeamento utilizado foi o mesmo do projeto CLIO [23]. Para os autores, a principal contribuição do trabalho é o desenvolvimento de dois algoritmos (*basic query rewriting algorithm* e *query resolution algorithm*). Segundo os autores, estes algoritmos foram os primeiros a tratar mapeamentos definidos diretamente sobre estruturas aninhadas. Além disso, consideram que os mapeamentos podem ser incompletos, ou seja, nem toda propriedade presente no *target schema* está presente em todas as fontes de dados. Em relação às restrições (restrições de integração), estas são definidas sobre o *target schema* e, de acordo com os algoritmos de [49], somente após a fase de consultas às bases é que estas restrições são consideradas para resolver conflitos e construir uma resposta semanticamente correta.

Neste trabalho, é definido um formalismo mais simples para a definição de mapeamentos sobre estruturas aninhadas. Este formalismo também permite definir restrições, porém diretamente entre as fontes de dados. Com estas restrições, o mediador pode gerar respostas semanticamente corretas e utilizar algumas estratégicas de otimização para melhorar o desempenho dos planos de execução de consultas gerados.

No contexto da integração de dados de fontes distribuídas e fragmentadas, algumas propostas [19][24] são apresentadas para o processamento eficiente de consultas. É importante observar que técnicas de fragmentação de dados são utilizadas com o objetivo de ganhos de desempenho no processamento de consultas. Em [24], é apresentado um método eficiente para o processamento de consultas distribuídas sobre grandes volumes de dados XML. Na estratégia seguida, considerando a estrutura e o tamanho dos dados, é realizada uma fragmentação vertical e uma distribuição destes fragmentos em múltiplos nodos. Assim, é possível balancear os custos de armazenamento e do processamento distribuído de consultas.

Em [19], é apresentada uma metodologia para o processamento de consultas XQuery sobre bases de dados XML distribuídas e fragmentadas. É utilizada a álgebra TLC[34] para a decomposição das consultas XQuery e a integração dos resultados. A técnica de fragmentação considerada provê regras formais para a reconstrução dos dados XML originais a partir de seus fragmentos. Esta metodologia pode ser utilizada para a integração de fontes que disponibilizam fragmentos homogêneos de dados XML, sem replicação de dados. Neste trabalho, propomos uma metodologia para o processamento de consultas sobre Visões de Mediação cujas etapas foram adaptadas da proposta de [19].

Capítulo III

Framework Proposto

Este capítulo apresenta maiores detalhes do *framework* proposto para sistemas de mediação baseados em XML. É também apresentada, através de um exemplo, uma visão geral do processo de geração de Visões de Mediação (VMs) e da metodologia para o processamento de consultas sobre VMs, os quais serão discutidos nos próximos capítulos.

3.1. A Arquitetura de Três Níveis de Esquemas

No Capítulo I, apresentamos sucintamente o *framework* proposto para sistemas de mediação baseados em XML. O *framework* adota uma arquitetura de três níveis de esquemas que objetiva simplificar e melhorar o desempenho do processamento de consultas sobre as visões de mediação. Nesta seção, voltamos a discutir, agora em mais detalhes, sobre os três níveis de esquemas desta arquitetura, exibidos na Figura 3.1(b).

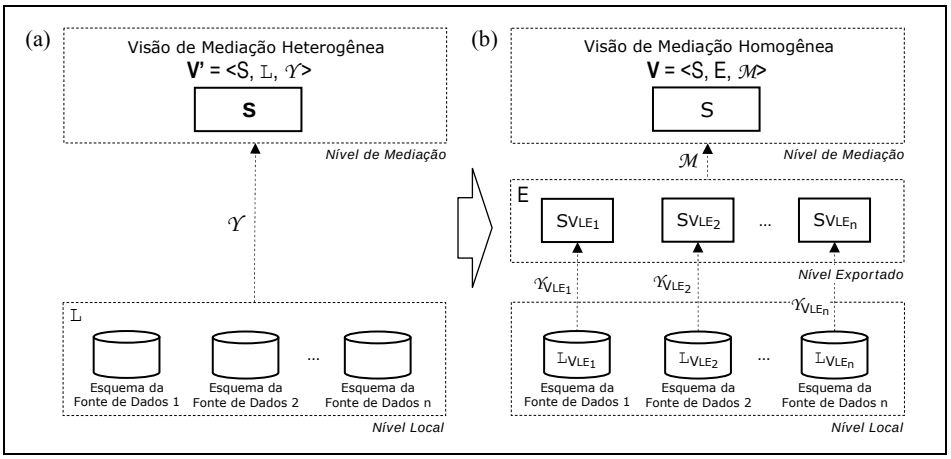


Figura 3.1 – (a) VM Heterogênea; (b) VM Homogênea (arquitetura de três níveis de esquema)

O *Nível Local* contém os esquemas de fontes de dados heterogêneas e distribuídas, onde os dados estão realmente armazenados.

O *Nível Exportado* contém um conjunto de visões XML chamadas de ***Visões Locais Exportadas* (VLEs)**, onde cada VLE é definida em termos do esquema de uma única fonte de dados local. A definição de uma VLE é dada por uma tripla $V_{LE} = <S_{VLE}, L_{VLE}, \mathcal{Y}_{VLE}>$, onde S_{VLE} é o esquema da visão, L_{VLE} é o esquema de uma fonte de dados e $\mathcal{Y}_{VLE}$ é o conjunto de assertivas que especificam o mapeamento entre S_{VLE} e L_{VLE}.

O *Nível de Mediação* contém um conjunto de visões XML chamadas de *Visões de Mediação* (VMs), onde cada VM é definida em termos de um conjunto de VLEs. As VLEs são fragmentos homogêneos das VMs e, por isso, estas VMs são também chamadas de *Visões de Mediação Homogêneas*. A definição de uma visão de mediação homogênea é dada por uma tripla **V = <S, E, $\mathcal{M}$ >**, onde S é o esquema da visão; E é o esquema exportado, que inclui as VLEs e restrições entre as VLEs e $\mathcal{M}$ é o conjunto de assertivas que especificam o mapeamento entre S e E.

Neste trabalho, uma VM homogênea é gerada a partir de uma visão de mediação equivalente especificada diretamente sobre os esquemas das fontes de dados heterogêneas, chamada de *Visão de Mediação Heterogênea*. Uma VM heterogênea é definida por uma tripla **V' = <S, L, $\mathcal{Y}$ >**, onde S é o esquema da visão, L é a união dos esquemas das fontes e restrições entre estes esquemas; e $\mathcal{Y}$ é o conjunto de assertivas que especificam o mapeamento entre S e L, conforme exibido na Figura 3.1(a).

No Capítulo V, VMs heterogêneas, VMs homogêneas e VLEs são definidas formalmente; são também apresentados em detalhes a abordagem para especificação de uma VM heterogênea e o processo de transformação desta especificação em uma VM homogênea definida em termos das VLEs.

Resumidamente, a especificação de uma VM heterogênea é realizada por um projetista em três passos: (i) é definido o esquema (S) da visão; (ii) são informadas as fontes (L) que deverão prover dados para compor a VM. Opcionalmente, são definidas restrições entre os esquemas destas fontes; e (iii) são definidos os mapeamentos da VM heterogênea em relação às fontes de dados ($\mathcal{Y}$). Os mapeamentos são definidos utilizando um formalismo proposto neste trabalho, chamado de Assertivas de Correspondência (ACs).

A transformação da especificação de uma VM heterogênea em uma VM homogênea, ou simplesmente, a geração de uma VM homogênea é realizada automaticamente, conforme se segue: (i) o esquema da VM heterogênea é atribuído como esquema (S) da VM homogênea; (ii) a partir dos mapeamentos da VM heterogênea é gerado o esquema exportado (E), ou seja, são geradas as VLEs e as restrições entre estas VLEs; e (iii) a VM homogênea a definida em termos das VLEs pelos seus mapeamentos ($\mathcal{M}$), os quais incluem uma expressão algébrica de reconstrução da VM homogênea a partir de operações de união e junção das VLEs que foram geradas.

3.2. O Framework Proposto

Na Figura 3.2 são mostrados os componentes do *framework* proposto. O processo de especificação e geração de VMs homogêneas (de acordo com a arquitetura de três níveis de esquemas) e a metodologia para o processamento de consultas sobre VMs homogêneas constituem as contribuições chave de nossa proposta.

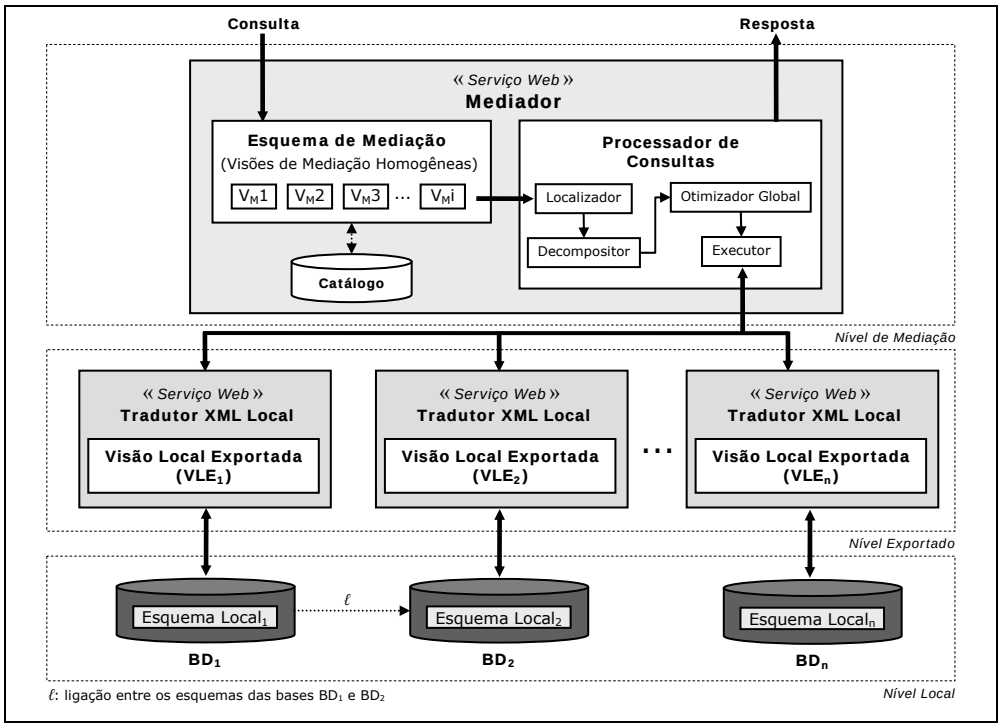


Figura 3.2 – *Framework* Proposto

Resumidamente, o processamento de uma consulta sobre uma VM homogênea é realizado conforme se segue: a consulta XQuery sobre a VM homogênea (nível de mediação) é decomposta, por um *Mediador*, em subconsultas XQuery sobre as VLEs (nível exportado). As subconsultas sobre as VLEs são traduzidas, por *Tradutores Locais*, em consultas sobre os esquemas das fontes de dados, utilizando linguagens locais de consulta (nível local). A partir dos resultados das subconsultas, o Mediador realiza a composição da resposta da consulta.

O *Mediador* é composto por:

(i) um *Processador de Consultas*, responsável pela implementação da metodologia para o processamento de consultas sobre as VMs;

(ii) um *Catálogo*, onde estão armazenadas as informações das VMs e VLEs geradas e qualquer outra informação necessária à execução do processamento de consultas. Várias VMs podem ser geradas e ter suas informações armazenadas no Catálogo, formando o *Esquema de Mediação*.

O Processador de Consultas é formado pelos componentes *Localizador*, *Decompositor*, *Otimizador Global* e *Executor*. Os três primeiros executam as etapas da metolodogia para o processamento de consultas proposta neste trabalho. Esta metodologia será apresentada resumidamente na próxima seção. O componente *Executor* é responsável pela execução do plano da consulta, conforme veremos a seguir.

O mediador e os tradutores locais consistem de serviços web que disponibilizam métodos de consulta às VMs e às VLEs, respectivamente. Para cada consulta sobre uma VM, o mediador gera e executa o *plano de execução da consulta*, representado por um *workflow* de processos, os quais se dividem em: (i) processos que invocam os serviços web locais, passando as subconsultas como parâmetro dos métodos de consulta às VLEs; e (ii) processos que realizam a composição da resposta da consulta a partir dos resultados obtidos nos serviços web locais.

No plano de execução, os processos responsáveis por invocar os serviços web locais armazenam como atributo as subconsultas que são enviadas para execução sobre as VLEs. Os demais processos realizam a composição dos resultados das subconsultas, gerando a resposta da consulta.

A implementação do mediador e dos tradutores locais como serviços web apresenta vantagens de interoperabilidade entre estes componentes e de flexibilidade do *framework* proposto, permitindo, por exemplo, que os tradutores sejam criados em diferentes linguagens de programação, executem em diferentes sistemas operacionais e sejam substituídos por outros mediadores, criando níveis hierárquicos de mediação.

Os serviços web que implementam o mediador e os tradutores locais disponibilizam o método *ConsultaVisaoXML*, que permite executar consultas XQuery sobre VMs ou VLEs, respectivamente. Os parâmetros de entrada e saída do método são apresentados na Figura 3.3.

Método:	ConsultaVisaoXML	
Entrada:		
V	string	Nome da VM ou da VLE a ser consultada
Q	string	Consulta XQuery para execução sobre V
Saída:		
R_Q	string	Resultado de Q (em XML)
status	boolean	Status da consulta (sucesso ou falha)

Figura 3.3 – Parâmetros do método ConsultaVisaoXML (Mediador e Tradutores Locais)

3.3. Metodologia para o Processamento de Consultas (Visão Geral)

Neste trabalho, propomos uma metodologia para o processamento consultas sobre VM homogêneas. Esta metodologia (e os algoritmos que a implementam) é apresentada em detalhes no Capítulo VI e está dividida em três etapas, as quais apresentamos brevemente a seguir:

1) ***Localização dos Dados***: são identificadas as VLEs que possuem informações relevantes para a resposta da consulta, ou seja, são identificadas as bases de dados que possuem informações relevantes;

2) ***Decomposição da Consulta***: a consulta sobre a VM é decomposta em subconsultas sobre as VLEs relevantes. As subconsultas são atribuídas aos processos do plano de execução responsáveis por enviá-las aos serviços web locais. É gerada a *expressão global de execução da consulta*, que consiste de uma expressão algébrica composta por operações de união e junção das subconsultas sobre as VLEs relevantes;

3) ***Otimização Global***: é gerado o plano de execução da consulta, ou simplesmente, o *workflow* de execução da consulta, que consiste uma estratégia de execução mais eficiente que a simples implementação da expressão global da consulta. Nesta etapa, são adotadas heurísticas que objetivam aumentar a eficiência do plano de execução. Por exemplo, é definida uma ordem de execução de forma que alguns processos possam ser executados em paralelo e não sejam utilizadas desnecessariamente operações de maior custo computacional. Além disso, todas as operações de junção seguem uma estratégia baseada na operação de semi-junção, objetivando, quando for o caso, reduzir o volume de dados trafegados das fontes de dados para o mediador.

Ao contrário de outros trabalhos [49][50], a decomposição da consulta pode ser realizada sem a necessidade de verificar, em tempo de execução, os mapeamentos entre o esquema da VM e os esquemas das fontes de dados. Conforme foi dito, esta característica decorre do fato das VLEs representarem fragmentos homogêneos da VM. Assim, é possível decompor uma consulta de forma simples apenas pela comparação das propriedades selecionadas na consulta com as propriedades disponíveis em cada VLE.

3.4. Exemplo

A seguir, é apresentado um breve exemplo da utilização do *framework* proposto. O exemplo contempla a especificação e a geração de uma VM homogênea e o processamento de uma consulta sobre esta VM. Maiores detalhes deste exemplo e dos conceitos, processos e algoritmos utilizados são apresentados nos capítulos seguintes.

Problema: Deseja-se integrar as informações de atendimentos clínicos realizados aos cidadãos de Fortaleza, registrados em hospitais e postos de saúde da rede pública. O objetivo é obter um "Prontuário Unificado" dos pacientes, possibilitando aos profissionais de saúde consultar, a qualquer momento, as patologias diagnosticadas e tratadas em cada paciente, independente do hospital ou posto de saúde onde foi realizado o atendimento.

3.4.1. Especificação e Geração da Visão de Mediação

- **Bases de Dados**

As bases de dados são previamente conhecidas pelo projetista e compartilham domínios comuns de informações. Suponha as bases de dados de um hospital (BD_1 – Hospital das Clínicas), de um posto de saúde (BD_2 – UBASF César Cals), do Ministério da Saúde (BD_3) e do Conselho Regional de Medicina – CE (BD_4). Na Figura 3.4, são exibidos os esquemas das bases de dados.

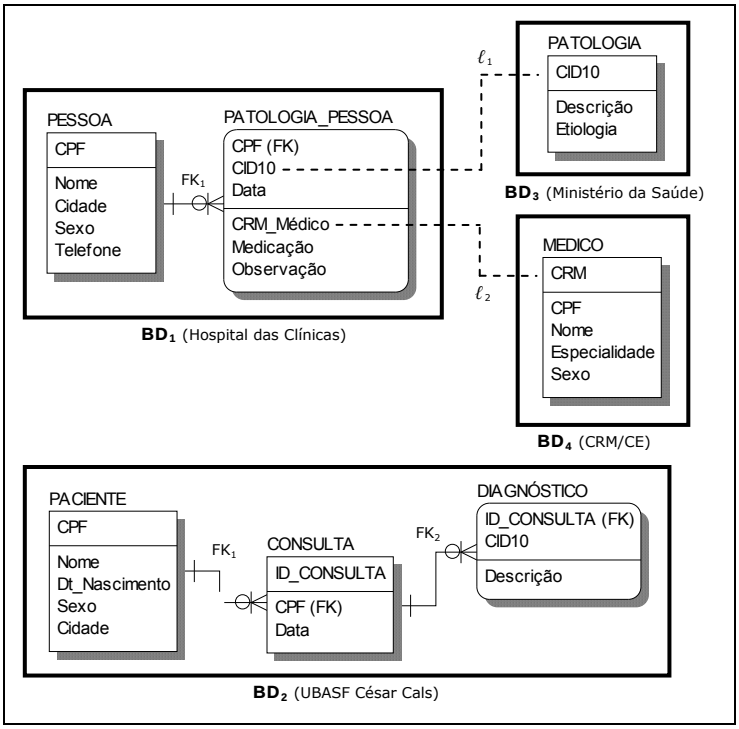


Figura 3.4 – Esquemas das bases de dados

Uma característica importante deste trabalho consiste em suportar a definição de *ligações* (virtuais) entre os esquemas de bases de dados distintas. Estas ligações são definidas pelo projetista e possibilitam navegar através dos esquemas das bases de dados e definir esquemas para as VMs mais ricos em informação.

Veremos que, na geração das VLEs, as ligações entre as bases de dados originam ligações entre as próprias VLEs que foram exportadas por aquelas bases. Assim, durante o processamento de uma consulta, propriedades de uma VLE podem ser complementadas por propriedades de outras VLEs, referenciadas por ligações, através de operações de junção.

A definição formal das ligações será vista no Capítulo IV. Observe na Figura 3.4 que o projetista definiu as ligações ℓ_1 e ℓ_2 entre o esquema de BD_1 e os esquemas de BD_3 e BD_4, respectivamente.

- **Especificação da VM Heterogênea e Geração da VM Homogênea**

Suponha que o projetista, visando atender à necessidade do "Prontuário Unificado", especificou uma VM heterogênea chamada $Paciente'_M$ e que foi transformada em uma VM homogênea equivalente chamada $Paciente_M$, cujo esquema $TPaciente_M$ é apresentado na Figura 3.5. Na especificação da VM heterogênea $Paciente'_M$, o projetista definiu assertivas de correspondência entre o esquema de $Paciente'_M$ e os esquemas das bases BD_1 e BD_2, as quais possuem informações de pacientes. As assertivas não são mostradas neste capítulo.

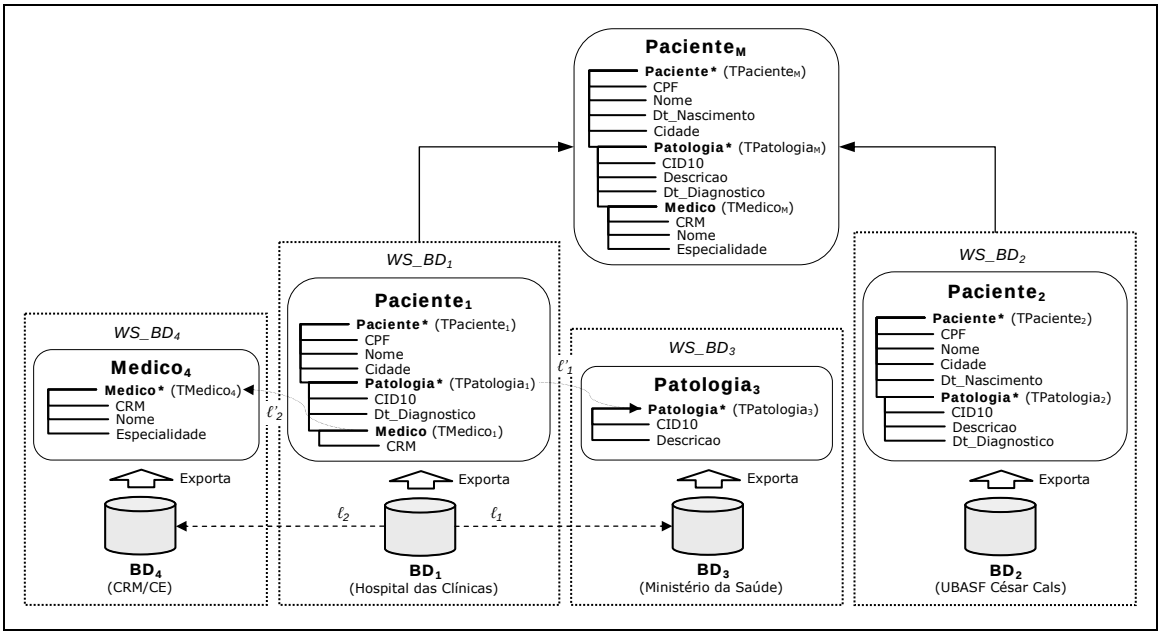


Figura 3.5 – VM $Paciente_M$ e VLEs $Paciente_1$, $Paciente_2$, $Patologia_3$ e $Medico_4$

Além da geração da VM homogênea Paciente_M, a partir da especificação de Paciente'_M foram também geradas as VLEs Paciente_1, Paciente_2, Patologia_3 e Medico_4, cujos esquemas são também apresentados na Figura 3.5. Estas VLEs são exportadas, respectivamente, pelos serviços web locais *WS_BD1*, *WS_BD2*, *WS_BD3* e *WS_BD4*.

As VLEs Paciente_1 e Paciente_2, que exportam dados das bases que possuem informações de pacientes, são fragmentos horizontais de Paciente_M. As VLEs Patologia_3 e Medico_4, que não exportam exatamente informações de pacientes, são fragmentos verticais de Paciente_M e servirão para complementar as informações da VLE Paciente_1.

Como as VLEs representam fragmentos da VM, o estado de Paciente_M pode ser definido por uma expressão algébrica formada por operações de união e junção dos estados das VLEs, conforme mostramos abaixo:

[Φ: Paciente$_M$ = (((Paciente$_1$ Ĵ Patologia$_3$) Ĵ Medico$_4$) Ü Paciente$_2$)], onde *Ĵ* e *Ü* representam operações algébricas de junção e união, respectivamente.

A expressão Φ é chamada de *expressão algébrica de reconstrução* de Paciente_M. Ela é gerada a partir do *Grafo da Hierarquia de VLEs* de Paciente_M, exibido na Figura 3.6. Neste grafo direcionado, o nó raiz representa a VM homogênea, os nós imediatamente abaixo do nó raiz representam as VLEs que são fragmentos horizontais (ou híbridos) e os demais nós representam as VLEs que são fragmentos verticais da VM. O grafo da hierarquia de VLEs e a expressão de reconstrução Φ são criados no final do processo de geração da VM homogênea Paciente_M e são armazenados no Catálogo do mediador. Eles serão utilizados no processamento de consultas.

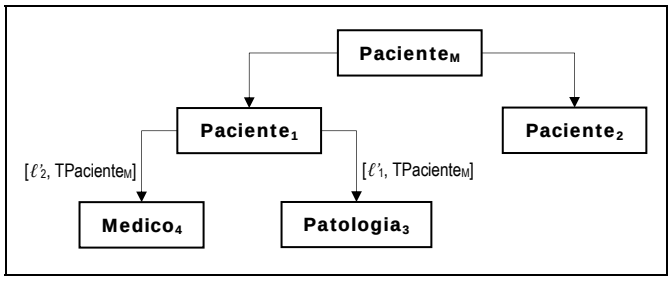


Figura 3.6 – Grafo da Hierarquia de VLEs de Paciente_M

Na definição das ACs de Paciente'_M com o esquema da base BD_1, o projetista utilizou as ligações ℓ_1 e ℓ_2 para definir assertivas com propriedades das bases BD_3 e BD_4. Por este motivo, foram geradas as VLEs Patologia_3 e Medico_4 para exportar dados daquelas bases. Note, portanto, que as ligações ℓ_1 e ℓ_2 entre as bases de dados originaram as ligações ℓ'_1 e ℓ'_2 (vide Figura 3.5) entre as respectivas VLEs geradas, ou seja:

- ℓ_1 originou ℓ'_1 entre $Paciente_1$ e $Patologia_3$;

- ℓ_2 originou ℓ'_2 entre $Paciente_1$ e $Medico_4$.

As operações de junção são possíveis pela existência das ligações ℓ'_1 e ℓ'_2, que determinam os elementos de junção (predicado de junção) das VLEs. Com a junção de $Paciente_1$ com $Patologia_3$ e $Medico_4$, as informações de Paciente/Patologia e Paciente/Patologia/Medico são complementadas com as propriedades Patologia/Descricao (de $Patologia_3$) e Medico/Nome e Medico/Especialidade (de $Medico_4$).

A possibilidade de utilizar ligações e gerar VLEs como $Patologia_3$ e $Medico_4$, que são fragmentos verticais da VM, é um diferencial importante deste trabalho, pois, como foi dito anteriormente, torna possível especificar VMs com uma maior riqueza de informações. Em [49], por exemplo, são consideradas para integração apenas bases de dados que contêm informações de entidades da mesma natureza da entidade do esquema global (*target schema*). Para o exemplo dado, apenas BD_1 e BD_2 seriam consideradas por [49], pois somente essas bases contêm informações de pacientes. Entretanto, para este trabalho, informações podem ser obtidas de BD_3 e BD_4 mesmo que estas bases não contenham exatamente informações de pacientes, mas, simplesmente, complementem as informações destes.

3.4.2. Processamento de Consultas

Nesta seção, são apresentadas as etapas do processamento de uma consulta $Q_{Paciente}$ sobre a VM $Paciente_M$, de acordo com a metodologia proposta neste trabalho.

A consulta $Q_{Paciente}$ é mostrada na Figura 3.7. Observe que nesta consulta são selecionados os pacientes da cidade de 'Fortaleza', através do filtro de seleção [Paciente/Cidade = 'Fortaleza'].

```
for $x1 in VIEW("PacienteM")/Paciente
where $x1/Cidade = 'Fortaleza'
return <paciente> {
          $x1/CPF,
          $x1/Nome,
          $x1/Dt_Nascimento,
          for $x2 in $x1/Patologia
          return <patologia>{
                    $x2/CID10,
                    $x2/Descricao,
                    $x2/Dt_Diagnostico
                  }</patologia>
      }</paciente>
```

Figura 3.7 – Consulta $Q_{Paciente}$ (XQuery)

Conforme foi dito anteriormente, o processamento de $Q_{Paciente}$ é realizado em três etapas, as quais serão resumidamente apresentadas a seguir:

1) Localização dos Dados;

2) Decomposição de $Q_{Paciente}$;

3) Otimização Global (Geração do Plano de Execução $Q_{Paciente}$)

- **Localização dos Dados**

Nesta etapa, são identificadas as VLEs que possuem informações relevantes para a consulta $Q_{Paciente}$. O resultado consiste de um grafo que contém apenas VLEs relevantes, chamado de *Grafo das VLEs relevantes* para $Q_{Paciente}$.

O grafo das VLEs relevantes é gerado a partir do grafo da hierarquia de VLEs de $Paciente_M$, eliminando-se os nós que representam VLEs que não são relevantes para $Q_{Paciente}$, conforme mostra a Figura 3.8. O grafo das VLEs relevantes é, portanto, um subgrafo do grafo da hierarquia de VLEs de $Paciente_M$.

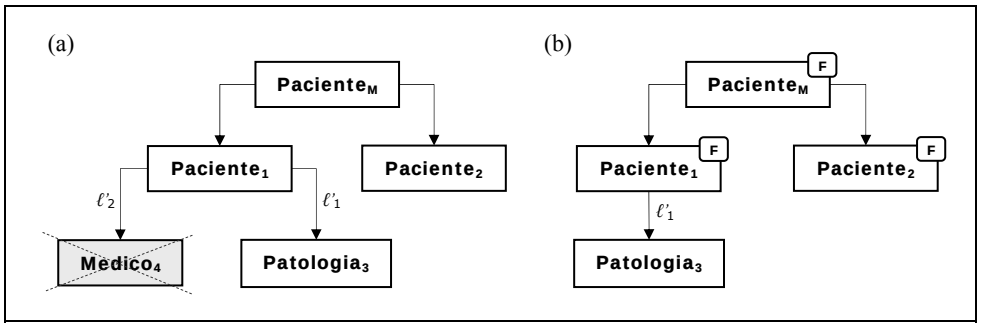


Figura 3.8 – (a) Grafo da Hierarquia de VLEs de $Paciente_M$; (b) Grafo das VLEs relevantes para $Q_{Paciente}$

Observe que, como a consulta $Q_{Paciente}$ não seleciona informações de médicos (propriedade Paciente/Patologia/Medico), a VLE $Medico_4$ é excluída do grafo de VLEs relevantes (vide Figura 3.8(a)).

No grafo das VLEs relevantes também são identificadas as VLEs que possuem propriedades sobre as quais foram aplicados os filtros de seleção em $Q_{Paciente}$. Os nós que representam estas VLEs (e todos os seus nós ancestrais) são assinalados com as *marcações de filtro* (F). Na geração do plano de execução de $Q_{Paciente}$, estas marcações serão utilizadas como um dos critérios para definir a ordem de execução das consultas sobre as VLEs. Observe novamente que, como as VLEs $Paciente_1$ e $Paciente_2$ possuem a propriedade Paciente/Cidade, sobre a qual foi aplicado o filtro [Paciente/Cidade = 'Fortaleza'], os nós que representam estas VLEs e todos os seus nós ancestrais são assinalados com as marcações de filtro (vide Figura 3.8(b)).

- **Decomposição de $Q_{Paciente}$**

Nesta etapa, a consulta $Q_{Paciente}$ é decomposta em subconsultas sobre as VLEs relevantes e é gerada a *expressão global de execução de* $Q_{Paciente}$, que consiste de uma expressão algébrica composta por operações de união e junção das subconsultas sobre as VLEs relevantes.

A expressão global é obtida a partir da expressão algébrica (Φ) de reconstrução da VM, removendo-se as VLEs irrelevantes e substituindo-se as VLEs relevantes pelas subconsultas correspondentes. O objetivo, portanto, consiste em reduzir a expressão algébrica de reconstrução da VM em uma expressão algébrica com subconsultas sobre as VLEs relevantes.

É importante ressaltar que a decomposição de $Q_{Paciente}$ é realizada apenas pela comparação das propriedades selecionadas em $Q_{Paciente}$ com as propriedades disponíveis nas VLEs. Os filtros aplicados sobre as propriedades em $Q_{Paciente}$ são refletidos nas subconsultas, desde que a VLE possua aquelas propriedades. Assim, na Figura 3.9 são exibidas as subconsultas Q_1, Q_3 e Q_2 que deverão ser executadas respectivamente sobre as VLEs $Paciente_1$, $Patologia_3$ e $Paciente_2$.

Seja, então, a expressão Φ de reconstrução de $Paciente_M$:

$[\Phi: Paciente_M \equiv ((Paciente_1 \hat{J} Patologia_3) \hat{J} Medico_4) \ddot{U} Paciente_2]$

A expressão global de execução de $Q_{Paciente}$ é dada por:

$[\Phi: Q_{Paciente} \equiv (Q_1 \hat{J} Q_3) \ddot{U} Q_2]$

Q_1

```
for $x1 in VIEW("Paciente1")/Paciente
where $x1/Cidade = 'Fortaleza'
return <paciente> {
          $x1/CPF,
          $x1/Nome,
          for $x2 in $x1/patologia
          return <patologia>{
                    $x2/CID10,
                    $x2/Dt_Diagnostico
                 }</patologia>
       }<paciente>
```

Q_3

```
for $x3 in VIEW("Patologia3")/Patologia
return <patologia> {
          $x3/CID10,
          $x3/Descricao
       }<patologia>
```

Q_2

```
for $x2 in VIEW("Paciente2")/Paciente
where $x2/Cidade = 'Fortaleza'
return <paciente> {
          $x2/CPF,
          $x2/Nome,
          $x2/Dt_Nascimento,
          <patologia>{
             $x2/CID10,
             $x2/Descricao,
             $x2/Dt_Diagnostico
          }</patologia>
       }<paciente>
```

Figura 3.9 – Subconsultas Q_1, Q_3 e Q_2

- **Otimização Global (Geração do Plano de Execução de $Q_{Paciente}$)**

Nesta etapa, é gerado o plano de execução de $Q_{Paciente}$, ou simplesmente, o *workflow* execução de $Q_{Paciente}$. A geração deste *workflow* adota heurísticas que objetivam aumentar o seu desempenho, como a redução do custo total das operações de integração e a redução do volume de dados trafegados das fontes de dados para o mediador.

O *workflow* de execução de $Q_{Paciente}$ é exibido na Figura 3.10. Observe que são gerados processos para: (i) consultar todas as VLEs do grafo de VLEs relevantes (processos PC); e (ii) combinar estes resultados (processos PJ e PO), considerando a hierarquia dos fragmentos horizontais (ou híbridos) e fragmentos verticais representados no grafo das VLEs relevantes.

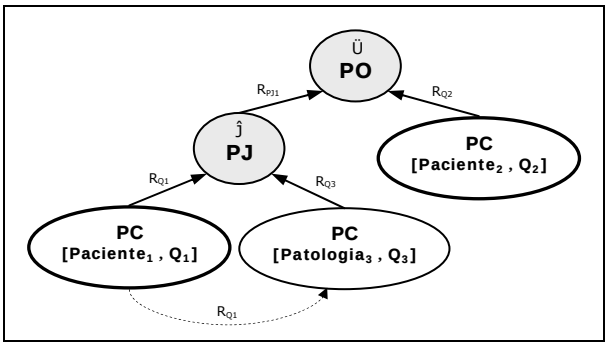


Figura 3.10 – *Workflow* de execução de $Q_{Paciente}$

As subconsultas geradas na etapa anterior são atribuídas aos processos PC, responsáveis por executá-las nos sites remotos através dos tradutores locais. Observe que o processo PC[Patologia$_3$, Q_3] depende do resultado do processo PC[Paciente$_1$, Q_1]. O objetivo desta dependência é possibilitar que, antes da junção (processo PJ), apenas dados complementares das patologias que pertencem ao resultado de PC[Paciente$_1$, Q_1] sejam consultados em Patologia$_3$.

A estratégia descrita acima está baseada na operação de semi-junção (*semijoin*) e objetiva reduzir o volume de dados que trafega entre o mediador e os serviços web locais. Para seguir esta estratégia, o algoritmo adiciona à subconsulta Q_3 uma cláusula *where*, que filtra o resultado de acordo com um conjunto de valores que será extraído do resultado do processo PC[Paciente$_1$, Q_1]. A subconsulta Q_3 passa a ser um consulta parametrizada (parâmetro *P3*), conforme exibido na Figura 3.11.

Q_3

```
for $x3 in VIEW("Patologia3")/Patologia
where ($x3/CID10 in (P3))
return <patologia> {
            $x3/CID10,
            $x3/Descrição
       }<patologia>
```

Figura 3.11 – Subconsultas Q_1, Q_3 e Q_2

Antes da execução de Q_3 pelo processo PC[Patologia$_3$, Q_3], a cláusula *where* é redefinida para filtrar os resultados conforme os valores obtidos de R_{Q1} (resultado de PC[Paciente$_1$, Q_1]), conforme se segue:

$P_3 = \{\$x \mid \$x \in R_{Q1}/Paciente/Patologia/CID10\}$

Desta forma, são buscadas de $Patologia_3$ apenas informações de patologias presentes no resultado de Q_1, reduzindo o volume de dados trafegados.

A seguir, apresentamos os processos do *workflow* de execução de $Q_{Paciente}$.

- Processo $PC[Paciente_1, Q_1]$: responsável pelo envio da consulta Q_1 ao serviço web local que exporta a VLE $Paciente_1$. A consulta Q_1 é executada pelo serviço web local e obtém as propriedades relevantes da VLE $Paciente_1$ para a resposta de $Q_{Paciente}$, de acordo com o filtro [Paciente/Cidade = 'Fortaleza']. O resultado R_{Q1} é o resultado do processo $PC[Paciente_1, Q_1]$.

- Processo $PC[Patologia_3, Q_3]$: responsável pelo envio da consulta Q_3 ao serviço web local que exporta a VLE $Patologia_3$. A consulta Q_3 é executada pelo serviço web local e obtém apenas informações complementares de patologias pertencentes ao resultado R_{Q1}. O resultado R_{Q3} é o resultado do processo $PC[Patologia_3, Q_3]$.

- Processo PJ: realiza a junção do resultado do processo $PC[Paciente_1, Q_1]$ com o resultado do processo $PC[Patologia_3, Q_3]$.

- Processo $PC[Paciente_2, Q_2]$: responsável pelo envio da consulta Q_2 ao serviço web local que exporta a VLE $Paciente_2$. A consulta Q_2 é executada pelo serviço web local e obtém as propriedades relevantes da VLE $Paciente_2$ para a resposta de $Q_{Paciente}$, de acordo com o filtro [Paciente/Cidade = 'Fortaleza']. O resultado R_{Q2} é o resultado do processo $PC[Paciente_2, Q_2]$.

- Processo PO: realiza a união externa (*Outer Union*) do resultado do processo PJ com o resultado do processo $PC[Paciente_2, Q_2]$.

Capítulo IV

Fundamentos Teóricos

Este capítulo fornece os fundamentos teóricos necessários à apresentação detalhada do processo de geração de VMs e da metodologia para o processamento de consultas. São apresentadas as terminologias, as operações de integração (baseadas em uma álgebra para XML) e os *workflows* utilizados neste trabalho, os quais implementam planos de execução de consultas.

4.1. Tipos e Coleções XML

- **Tipos XML**

Neste trabalho, são utilizados somente os tipos XML simples definidos na especificação do XML Schema [47] e tipos XML complexos *restritos*. Um tipo XML complexo é *restrito* sse são utilizados em sua definição apenas os construtores *complexType* e *sequence* do XML Schema, e o tipo de seus atributos é um tipo XML simples.

A representação gráfica utilizada neste trabalho para um tipo XML complexo restrito é dada por uma estrutura em "árvore de diretórios", onde os nós representam elementos ou atributos. Caso o elemento representado por um nó seja de tipo complexo, toda a sua estrutura é representada como uma subárvore abaixo deste nó. Os nós são rotulados com o nome do elemento ou atributo, o tipo XML e, se for o caso, a restrição de ocorrência "*" para elementos de múltipla ocorrência. Atributos têm seu nome precedido do símbolo "@".

(a)

```
<complexType name="TPaciente">
  <sequence>
     <element name="CPF" type="string"/>
     <element name="Nome" type="string"/>
     <element name="Dt_Nascimento" type="date"/>
     <element name="Cidade" type="string"/>
     <element name="Patologia" type="TPatologia"
              maxOccurs="unbounded">
  </sequence>
</complexType>
<complexType name="TPatologia">
  <sequence>
     <element name="CID10" type="string"/>
     <element name="Dt_Diagnostico" type="date"/>
  </sequence>
</complexType>
```

(b)

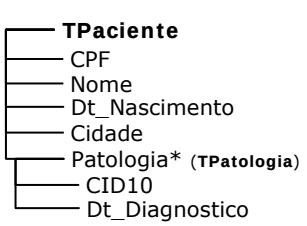


Figura 4.1 – (a) Declaração de TPaciente e TPatologia ; (b) Representação gráfica de TPaciente

Na Figura 4.1(a) é exibida a declaração dos tipos XML complexos restritos TPaciente e TPatologia. Na Figura 4.1(b) é exibida a representação gráfica destes tipos. Observe que o primeiro nó da árvore é o nome do tipo. Observe ainda que, no caso do elemento patologia, como este é do tipo complexo TPatologia, sua estrutura é representada como uma subárvore, abaixo do nó que representa este elemento.

- **Coleção XML**

Neste trabalho, uma *coleção XML* é uma tupla **C**=<e, T_e> onde e é o nome dos elementos primários da coleção **C** e T_e é o tipo de e. O tipo T_e é um tipo XML complexo restrito. O par <e, T_e> é também chamado de *Esquema* de **C**.

- *Estado* de **C**: é um conjunto não-ordenado de elementos e do tipo T_e. O estado corrente de uma coleção XML **C** é denotado por **$C.**

- *Chave* de **C**: conjunto de atributos ou elementos de tipo simples de T_e cujos valores identificam unicamente um elemento e em um estado de **C**.

No restante deste trabalho, T_e pode ser generalizado como o tipo da coleção **C**. O estado de **C** também pode ser representado graficamente por uma estrutura em "árvore de diretórios", semelhante à representação dos tipos XML. Entretanto, os nós da árvore que representam atributos ou elementos de tipos simples são rotulados pela concatenação do nome do elemento/atributo e pelo seu respectivo valor.

Por exemplo, seja a coleção **Pacientes**=<paciente, TPaciente>. Na Figura 4.2(a) é exibido um exemplo do estado de **Pacientes**, ou seja, um conjunto de elementos <paciente> de tipo TPaciente. Na Figura 4.2(b) é exibida a representação gráfica do estado de **Pacientes**. O elemento <CPF> pode ser considerado uma chave de **Pacientes**.

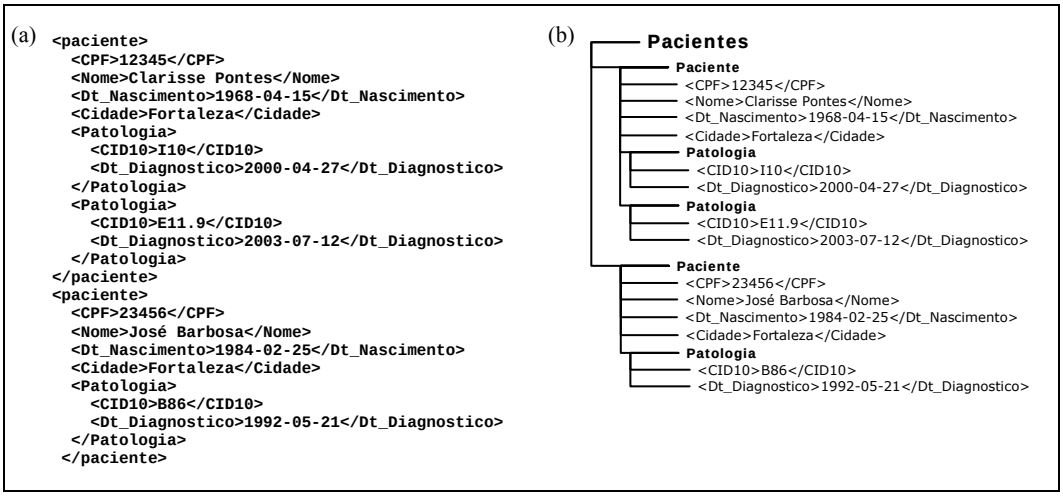


Figura 4.2 – (a) Estado da coleção Pacientes ; (b) Representação gráfica do estado de Pacientes

- **Restrição de Interseção entre Coleções XML**

Sejam $\mathbf{C_1}$ e $\mathbf{C_2}$ coleções XML. Diz-se que $\mathbf{C_1}$ e $\mathbf{C_2}$ *estão em interseção* sse $\mathbf{\$C_1}$ e $\mathbf{\$C_2}$ possuem elementos primários com valores de chaves iguais. A *restrição de interseção* entre $\mathbf{C_1}$ e $\mathbf{C_2}$ é denotada por $[\mathbf{C_1} \cap \mathbf{C_2} \neq \emptyset]$.

- **Expressão de Caminho**

Uma *expressão de caminho* ou um *caminho* é utilizado para localizar determinado nó em uma árvore. A expressão retorna uma seqüência de nós distintos e correspondentes ao nó localizado, na ordem em que se apresentam na árvore. Uma expressão de caminho consiste de um ou mais passos, onde cada passo representa um movimento ao longo dos nós, seja ascendente ou descendente. Cada passo retorna uma lista de nós que servem como ponto de partida para o passo seguinte.

Considere T_1, ..., T_n tipos XML. Suponha que T_k tem uma propriedade p_k de tipo T_{k+1}, para $k=1, ..., n-1$. Assim, é dado que:

1) $\delta = p_1 / p_2 / ... / p_{n-1}$ é um *caminho* de T_1;
2) T_n é o tipo de δ; e
3) δ tem *ocorrência simples* sse p_k tem ocorrência simples, para $k=1, ..., n-1$; caso contrário, δ tem *ocorrência múltipla*.

Seja, por exemplo, um elemento **$p** da coleção **Pacientes**. O caminho **$p/patologia/descricao** retorna os elementos <descricao>, que são subelementos dos elementos <patologia>, que são subelementos de **$p**.

- **Coleções Aninhadas**

Seja **C** uma coleção XML cujo esquema é <e,T> e $\delta = e_1 /.../ e_n$ um caminho de ocorrência múltipla de T. **C**(δ) denota uma *coleção aninhada de* **C**, de esquema $\langle e_n, T_{e_n} \rangle$, e cujo estado corrente é definido por: { **$p** | $\exists$**$c**$\in$**$C** tal que **$p**$\in$**$c**/δ }.

```
<Patologia>
  <CID10>I10</CID10>
  <Dt_Diagnostico>2000-04-27</Dt_Diagnostico>
</Patologia>
<Patologia>
  <CID10>E11.9</CID10>
  <Dt_Diagnostico>2003-07-12</Dt_Diagnostico>
</Patologia>
<Patologia>
  <CID10>B86</CID10>
  <Dt_Diagnostico>1992-05-21</Dt_Diagnostico>
</Patologia>
```

Figura 4.3 – Estado da coleção aninhada Pacientes(Patologia)

Seja, por exemplo, o estado corrente da coleção **Pacientes** exibido na Figura 4.2. **Pacientes**(patologia) é uma coleção aninhada de **Pacientes**, cujo esquema é dado por <patologia, TPatologia> e o estado corrente é exibido na Figura 4.3.

- **Ligações entre Coleções**

Uma ligação relaciona elementos de duas coleções, através de uma função de mapeamento entre estes elementos, conforme definido a seguir.

- Função de mapeamento: Seja $\mathbf{C_1}$ e $\mathbf{C_2}$ coleções de esquemas $<e_1,T_1>$ e $<e_2,T_2>$, respectivamente. Seja ainda:

  - $a_1, ..., a_n$ propriedades de T_1; e

  - $b_1, ..., b_n$ propriedades de T_2.

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

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

Uma *ligação de* $\mathbf{C_1}$ *para* $\mathbf{C_2}$ *através de f*, denotada por ℓ: $\mathbf{C_1} \xrightarrow[f]{} \mathbf{C_2}$, onde ℓ é o nome da ligação, ocorre quando $\{a_1, ..., a_n\}$ é chave de $\mathbf{C_1}$ e $\{b_1, ..., b_n\}$ é chave de $\mathbf{C_2}$. Neste caso, f define um mapeamento 1-1 dos elementos de $\mathbf{C_1}$ em elementos de $\mathbf{C_2}$.

Para uma elemento primário $\mathbf{\$c_1}$ de $\mathbf{\$C_1}$, a expressão de caminho $\mathbf{\$c_1}/\ell$ retorna o elemento $\mathbf{\$c_2}$ de $\mathbf{\$C_2}$ tal que $\mathbf{\$c_2}$ é referenciado pelo elemento $\mathbf{\$c_1}$ através de f, ou seja,

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

No decorrer deste trabalho, utilizaremos uma notação simplificada para representar ligações entre coleções. Assim, a mesma *ligação de* $\mathbf{C_1}$ *para* $\mathbf{C_2}$ *através de f* utilizando a notação simplificada será denotada por:

ℓ : $<\mathbf{C_1}, \{a_1, ..., a_n\}, \mathbf{C_2}, \{b_1, ..., b_n\}>$

Implicitamente, a função de mapeamento considerada é f:$T_1\{a_1, ..., a_n\} \rightarrow T_2\{b_1, ..., b_n\}$, onde T_1 e T_2 são os tipos de C_1 e C_2, e $\{a_1, ..., a_n\}$ e $\{b_1, ..., b_n\}$ são as chaves de C_1 e C_2.

Por exemplo, seja o estado corrente da coleção **Patologias,** exibido na Figura 4.4(a), cujos elementos <patologia> são do tipo TPatologia', exibido na Figura 4.4(b). Seja ainda a coleção aninhada **Pacientes**(patologia) exibida anteriormente na Figura 4.3 e uma função de mapeamento f:TPatologia{CID10}$\rightarrow$TPatologia'{CID10}, onde {CID10} é chave das coleções **Pacientes**(patologia) e **Patologias**.

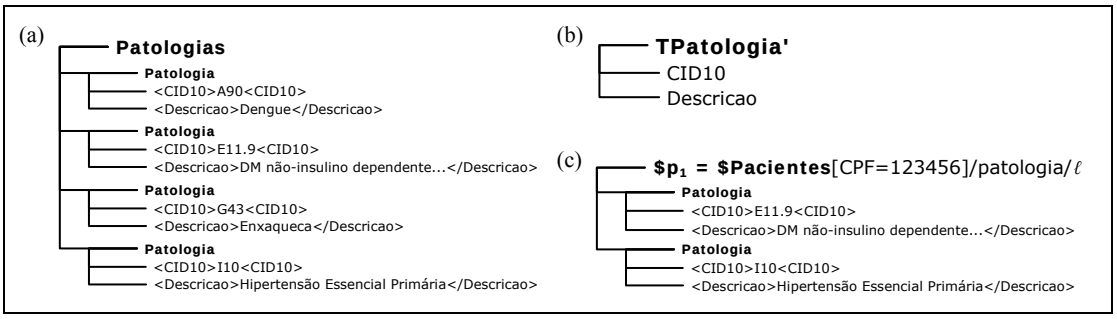


Figura 4.4 – Estado da coleção aninhada Pacientes(Patologia)

Assim, ℓ: **Pacientes**(patologia) $\xrightarrow{f}$ **Patologias** (ou ℓ:<**Pacientes**(patologia),{CID10}, **Patologias**,{CID10}>) é uma ligação da coleção aninhada **Pacientes**(patologia) para a coleção **Patologias** através de *f*. Em relação aos elementos de **Pacientes**(patologia), considere, por exemplo, o elemento $\mathbf{\$p_1}$ em **$Pacientes**, tal que $\mathbf{\$p_1}$ = **$Pacientes**[CPF=12345]. A expressão $\mathbf{\$p_1}$/patologia/$\ell$ retorna a coleção XML exibida na Figura 4.4(c). Observe que podem existir patologias na coleção **Pacientes**(patologia) que não têm correspondência em **Patologias**. É o caso do elemento $\mathbf{\$p_2}$ = **$Pacientes**/patologia[CID10=B86], o qual não tem correspondente em $\mathbf{\$p_1}$/patologia/$\ell$.

4.2. Integração de Coleções XML

Neste trabalho, a integração de coleções XML é realizada utilizando um conjunto de operadores da álgebra *Tree Logical Class* – TLC [34]. Embora não exista um consenso sobre uma álgebra para consulta e manipulação de dados XML, os autores da TLC destacam diversas vantagens da sua proposta em relação a outros trabalhos, especialmente através de experimentos.

Maiores detalhes da álgebra TLC podem ser obtidos em [34]. A seguir, apresentamos um resumo dos conceitos básicos desta álgebra para que possamos apresentar os operadores utilizados.

4.2.1. A Álgebra TLC

Álgebras são utilizadas para prover um fundamento formal a operações de consulta e manipulação de dados, possibilitando inclusive a implementação e a otimização destas operações. Devido à complexidade inerente ao modelo semi-estruturado, ainda não há uma álgebra de consenso para a consulta e a manipulação de dados XML. Muitos trabalhos têm sido propostos, entre eles o que apresenta a álgebra *Tree Logical Class* – TLC [34], onde os seus autores destacam diversas vantagens sobre outras propostas.

A álgebra TLC, implementada no banco de dados nativo XML Timber [41], foi derivada da álgebra *Tree Algebra for XML* – TAX [26]. A TAX opera sobre coleções de árvores (árvores de dados XML) utilizando uma álgebra semelhante à utilizada no modelo relacional. Cada operador da TAX recebe uma ou mais coleções de árvores como entrada e produz uma coleção de árvores como saída.

Diferentemente de uma coleção de tuplas, uma coleção de árvores XML pode possuir árvores com estruturas heterogêneas. Assim, surge a necessidade de criação de alguma estratégia que diminua essa heterogeneidade e se possa tratar de maneira uniforme as árvores de uma coleção. Para isto, a TAX apresenta o conceito de ***Padrão de Árvore*** (*Pattern Tree*), que permite identificar um subconjunto de nós de interesse em uma árvore e definir predicados de seleção em qualquer nó desta árvore. Desta forma, é possível se obter uma padronização das árvores de uma coleção de árvores heterogêneas e manipulá-las nas operações da álgebra. O conceito de Padrão de Árvore possibilita definir grande parte dos operadores semelhantes aos existentes na álgebra relacional (seleção, projeção, junção, agrupamento e ordenação), com algumas modificações apropriadas para o modelo XML.

A álgebra TLC estendeu o conceito de padrão de árvore para ***Padrão de Árvore Anotado*** (*Annotated Pattern Tree - APT*). As anotações se referem à cardinalidade dos nós da árvore de saída quando o APT for aplicado a uma árvore de entrada (casamento de nós da árvore de entrada com o APT). Assim, é possível se obter árvores de saída com estruturas heterogêneas, superando uma limitação da TAX, onde todas as árvores de saída possuem estrutura homogênea ao padrão de árvore. Além disso, a TLC cria o conceito de ***Classes Lógicas*** (*logical classes - LC*) e seus ***rótulos*** (*logical class labels - LCL*) para identificar mais facilmente conjuntos de nós nas árvores de saída e utilizá-los como entrada de outras operações.

Na Figura 4.5 são exibidas as possíveis cardinalidades para um nó de um APT; na Figura 4.6 apresentamos um exemplo de um APT e na Figura 4.7 é mostrado um exemplo de aplicação de um APT em três árvores de entrada, rotuladas por Paciente_1, Paciente_2 e Paciente_3, gerando três árvores de saída.

Para produzir árvores de saída, as árvores de entrada (vide Figura 4.7(a)) devem ser compatíveis com o APT (vide Figura 4.7(b)). Por exemplo, a árvore de entrada Paciente_3 não possui o elemento <Nome> filho de <Paciente>, como especificado pelo APT. Neste caso, nenhuma árvore de saída é gerada. Por outro lado, pode-se observar na Figura 4.7(c), que a árvore Paciente_2 gerou duas árvores de saída, pois contém dois

elementos <Patologia> filhos de <Paciente> e o APT especifica que na saída deve existir zero ou um elemento <Patologia> filho de <Paciente> (anotação "?"). Em relação às classes lógicas, cada nó do APT representa uma classe lógica e os nós da árvore de saída também são rotulados de acordo com a classe lógica a qual pertencem. Assim, no APT deste exemplo, temos seis classes lógicas: Paciente, CPF, Nome, Patologia, CID10 e DtDiagnostico. Os nós <CPF_1> e <CPF_2> das árvores de saída pertencem à classe lógica CPF. Os rótulos (1), (2), (3), (4), (5) e (6) identificam unicamente as classes lógicas e podem ser utilizados como referências para os nós destas classes dentro de operações algébricas.

-	um e somente um elemento permitido no casamento com a árvore de entrada
?	zero ou um elemento permitido no casamento com a árvore de entrada
+	um ou mais elementos permitidos no casamento com a árvore de entrada
*	zero ou mais elementos permitidos no casamento com a árvore de entrada

Figura 4.5 – Cardinalidade dos elementos de um APT

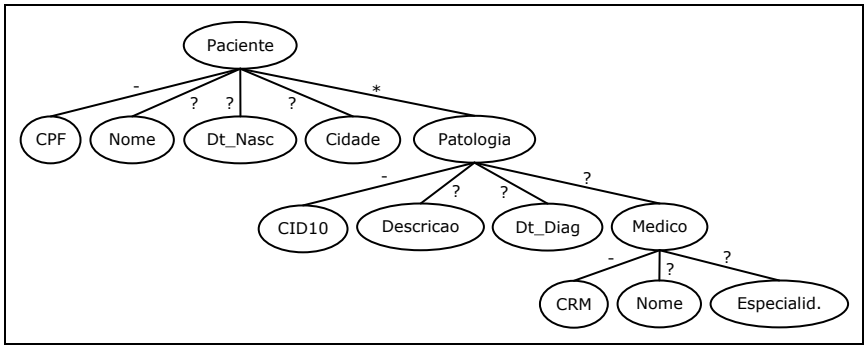


Figura 4.6 – Exemplo de um Padrão de Árvore Anotado (APT)

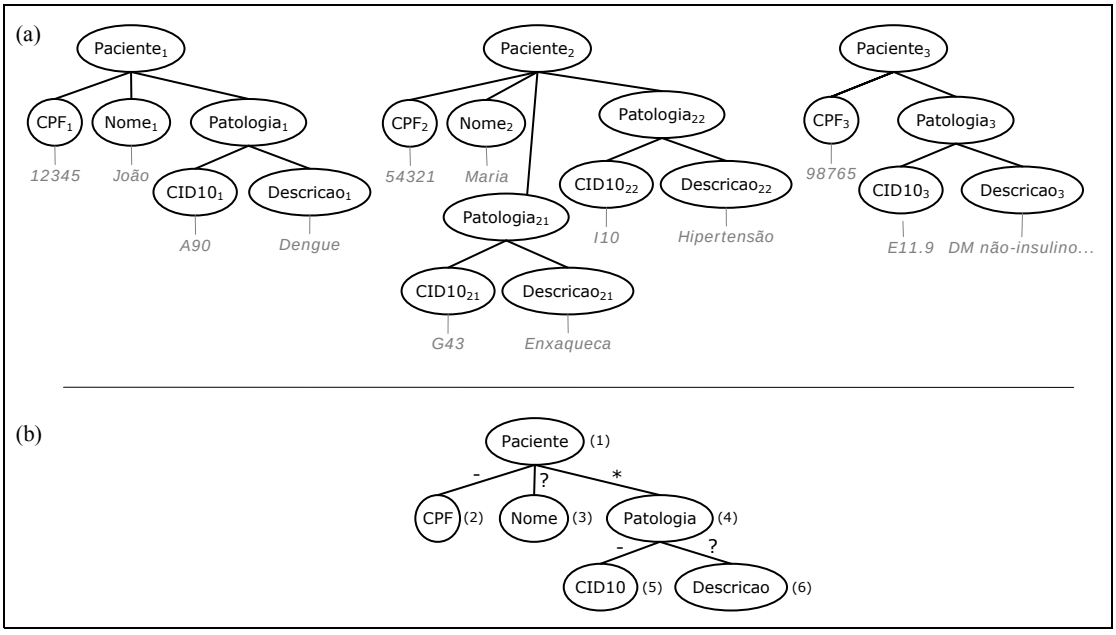


Figura 4.7 – Exemplo de aplicação de um APT. (a) Árvores de Entrada; (b) APT;

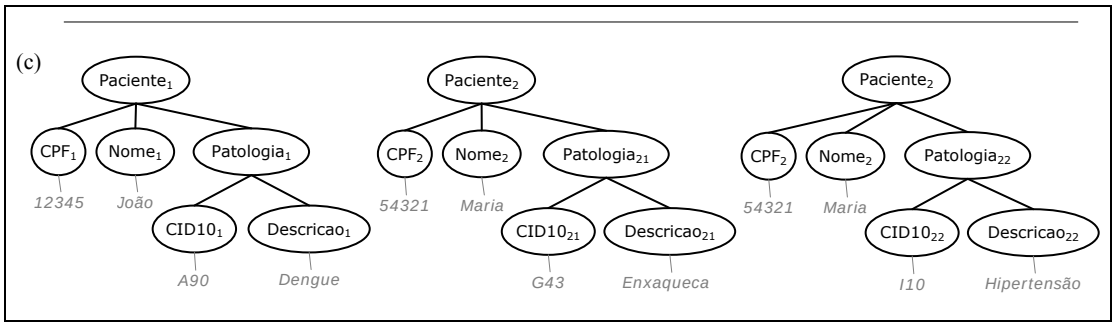


Figura 4.7 – (Continuação) (c) Árvores de Saída.

4.2.2. Operadores de Integração

Além de definir os conceitos de APTs, classes lógicas e rótulos de classes lógicas, a TLC define um conjunto de operações algébricas que recebem como entrada um ou mais conjuntos de árvores e produzem como saída um outro conjunto de árvores. Semelhante ao modelo relacional, as operações da TLC formam um modelo fechado, onde árvores de saída de uma operação podem servir de entrada para as outras operações. Desta forma, é possível definir expressões algébricas.

As principais operações algébricas da TLC são: filtro, junção (e algumas derivações), seleção, projeção, funções de agregação e construção. Uma lista mais completa de operações pode ser encontrada em [34].

Neste trabalho, utilizaremos essencialmente um subconjunto das operações da TLC relacionadas com a união e junção de coleções de dados XML. As coleções que serão integradas por estas operações representam fragmentos homogêneos da VM, uma vez que são originadas da execução de subconsultas sobre as VLEs (fragmentos homogêneos da VM) ou da execução de uma operação algébrica anterior (caso das expressões algébricas).

Algumas operações da TLC requerem um APT para determinar quais árvores de entrada serão manipuladas ou determinar a estrutura das árvores de saída. Como existe um esquema único e conhecido para cada VM, as estruturas dos APTs utilizados nas operações de integração serão trivialmente inferidas a partir do esquema da VM (ou do esquema de um elemento da VM). Por exemplo, a estrutura do APT da Figura 4.6 pode ser trivialmente inferida do tipo TPaciente da VM homogênea Paciente_M, apresentada no Capítulo III.

A seguir apresentamos as operações da TLC utilizadas neste trabalho.

- ***Select*** **(*S*)**

O operador *Select S[apt](S)* recebe como entrada um conjunto de árvores S (coleções XML) e um padrão de árvore anotado apt. Para cada árvore de entrada de S é realizado o casamento (*matching*) com o apt, gerando como resultado um conjunto de árvores que satisfazem o apt.

- ***Project*** ***(P)***

O operador *Project P[nl](S)* aceita como entrada um conjunto de árvores S e uma lista nl composta por rótulos de classes lógicas (LCL) para identificar conjuntos de nós. Para cada árvore de entrada de S, apenas os nós identificados em nl são mantidos no resultado da operação.

- ***Duplicate Elimination*** ***(DE)***

O operador *Duplicate Elimination DE[nl, ci](S)* realiza a eliminação de duplicatas em um conjunto de árvores S baseado na lista de nós especificados em nl. A lista nl é composta por rótulos de classes lógicas que correspondem a um conjunto único de nós das árvores de S. O parâmetro ci especifica se as duplicatas são eliminadas com base no conteúdo ou identificador do nó.

- ***Construct*** **(*C*)**

O operador *Construct C[c](S)* aplica um padrão de árvore-de-construção anotado c sobre um conjunto de árvores S e gera um conjunto de árvores de saída de acordo com c. Um padrão de árvore-de-construção anotado é um APT com algumas facilidades para construção arbitrária de árvores de saída.

- ***Union*** **(*U*)**

O operador *Union U[apt](S_l, S_r)* realiza a união de dois conjuntos de árvores (coleções XML) de entrada S_l e S_r, onde a estrutura da árvore resultante é definida por um padrão de árvore anotado *apt*.

Neste trabalho, utilizamos a notação simplificada **(S_l U S_r)** para representar uma expressão algébrica TLC que realiza a união de dois conjuntos de árvores S_l e S_r, baseada no operador *Union*.

Sejam s_l e s_r as classes lógicas que representam os nós-raiz das árvores de S_l e S_r, respectivamente. A expressão algébrica TLC correspondente a (S_l U S_r) é apresentada na Figura 4.8.

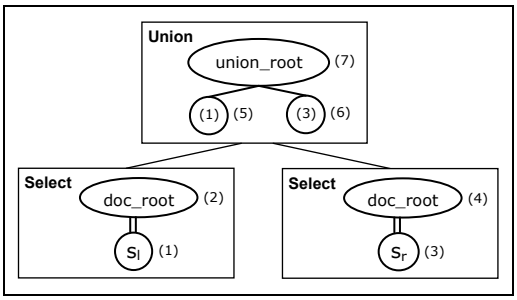


Figura 4.8 – Expressão algébrica representada por (S_l U S_r)

- ***Left-Outer-Nest-Value-Join* (Ĵ)**

O operador *Left-Outer-Nest-Value-Join* $LONVJ[LC_l, p, LC_r](S_l, S_r)$ recebe como entrada dois conjuntos de árvores (coleções XML) S_l e S_r ; dois rótulos de classes lógicas LC_l e LC_r (um para cada conjunto de árvores); e um predicado p entre LC_l e LC_r. O rótulo LC_l deve corresponder a um conjunto único de nós para cada árvore de S_l e o rótulo LC_r deve corresponder aos nós-raiz das árvores de S_r. O resultado do operador consiste da junção de cada árvore de S_l com todas as árvores de S_r que satisfazem o predicado p (critério de junção). Além disso, as árvores de S_l que não casam com árvores de S_r são também mantidas do resultado pela condição *left-outer* do operador.

Utilizamos a notação simplificada $(S_l \hat{J}_p S_r)$ para representar uma expressão algébrica TLC que realiza a junção de dois conjuntos de árvores S_l e S_r, baseada no operador *Left-Outer-Nest-Value-Join*. O parâmetro p é um predicado (critério de junção) entre nós das árvores de S_l e S_r. O predicado p é da forma $[s_l/\delta_i = s_r/\delta_j]$, onde s_l e s_r são classes lógicas que representam os nós-raiz das árvores de S_l e S_r, e δ_i e δ_j são caminhos de s_l e s_r, respectivamente.

Resolução de Conflitos

Neste trabalho, consideramos que as fontes de dados podem conter, simultaneamente, informações de uma mesma entidade do mundo real (*overlapping* ou interseção de informações). Assim, para gerar resultados semanticamente corretos (de acordo com o esquema de uma Visão de Mediação), pode ser necessário resolver conflitos [9] durante a integração de dados oriundos destas fontes. Este é o caso, por exemplo, das operações de junção. Eventualmente, estas operações geram resultados com nós repetidos de uma mesma classe lógica, mas o resultado semanticamente correto deveria conter apenas um único nó desta classe.

Propomos, então, a notação simplificada $(S_l \hat{J}_{T,p} S_r)$ para representar uma expressão algébrica TLC que realiza a junção, baseada no operador *Left-Outer-Nest-*

Value-Join, de dois conjuntos de árvores S_l e S_r e, em seguida, realiza uma operação de *Construct* (C) para resolver conflitos e gerar árvores de saída de acordo com um padrão de árvore-de-construção anotado c. O parâmetro T é o tipo XML de uma Visão de Mediação e é utilizado como base (modelo) para a geração de c. Para resolver conflitos, respeitando-se a estrutura e a cardinalidade dos elementos de T, são incluídos em c: (i) nós com referências para todas as classes lógicas de S_l ; e (ii) nós com referências apenas para as classes lógicas de S_r que não pertencem a S_l ou que correspondem a elementos de ocorrência múltipla em T. Portanto, os conflitos de nós repetidos de uma mesma classe lógica são resolvidos decidindo-se pelos dados oriundos do operando S_l.

Sejam s_l e s_r as classes lógicas que representam os nós-raiz das árvores de S_l e S_r, respectivamente. Seja o predicado p dado por $[s_l/n_1/.../n_{i-1}/n_i = s_r/m_1/.../m_{j-1}/m_j]$. A expressão algébrica TLC correspondente a $(S_l \hat{J}_{T,p} S_r)$ é apresentada na Figura 4.9. O padrão de árvore-de-construção anotado c da operação *Construct* é gerado dinamicamente durante o processamento de uma consulta, baseado em T e nas classes lógicas de S_l e S_r. Por isso, adicionamos simbolicamente T na representação desta operação.

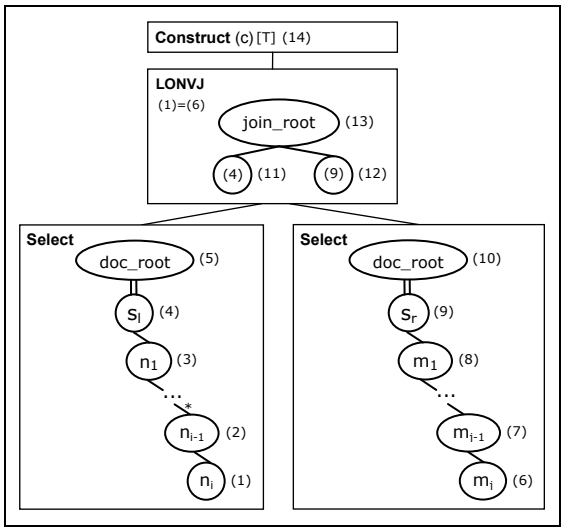


Figura 4.9 – Expressão algébrica representada por $(S_l \hat{J}_{T,p} S_r)$

- ***Outer Union* (Ü)**

O operador *Outer Union* (união externa) não é definido pela TLC. Entretanto, com o objetivo de simplificar notações e algoritmos deste trabalho, definimos esta operação através de uma composição de operadores da TLC.

De acordo com [17], uma operação de *Outer Union* no modelo relacional é equivalente a uma operação de *full outer join* (junção externa total) entre duas relações. Assim, para obtermos um resultado equivalente a uma operação de *Outer Union* (ou *full*

outer join) entre dois conjuntos de árvores S_l e S_r, propomos a seguinte composição de operações da TLC:

1) Utilização do operador *Left-Outer-Nest-Value-Join* para realizar a junção de S_l com S_r;
2) Utilização do operador *Left-Outer-Nest-Value-Join* para realizar a junção de S_r com S_l (sentido inverso da junção);
3) União dos resultados dos passos 1 e 2;
4) Eliminação de árvores duplicadas (*Duplicate Elimination*) do resultado do passo 3.

Utilizamos a notação simplificada $(S_l \ddot{U}_p S_r)$ para representar uma expressão algébrica TLC que realiza a união externa de dois conjuntos de árvores S_l e S_r, baseada no operador *Outer Union*. O parâmetro p é um predicado (critério de junção) entre nós das árvores de S_l e S_r, cuja forma foi mostrada anteriormente.

De forma semelhante ao operador *Left-Outer-Nest-Value-Join*, para resolver conflitos de dados oriundos de fontes que podem conter informações em interseção, utilizamos a notação simplificada $(S_l \ddot{U}_{T,p} S_r)$ para representar uma expressão algébrica TLC que realiza a *Outer Union* de dois conjuntos de árvores S_l e S_r e, em seguida, realiza uma operação de *Construct* (*C*) para resolver conflitos e gerar árvores de saída de acordo com um padrão de árvore-de-construção anotado c. O parâmetro T é o tipo XML de uma Visão de Mediação e é utilizado como base (modelo) para a geração de c.

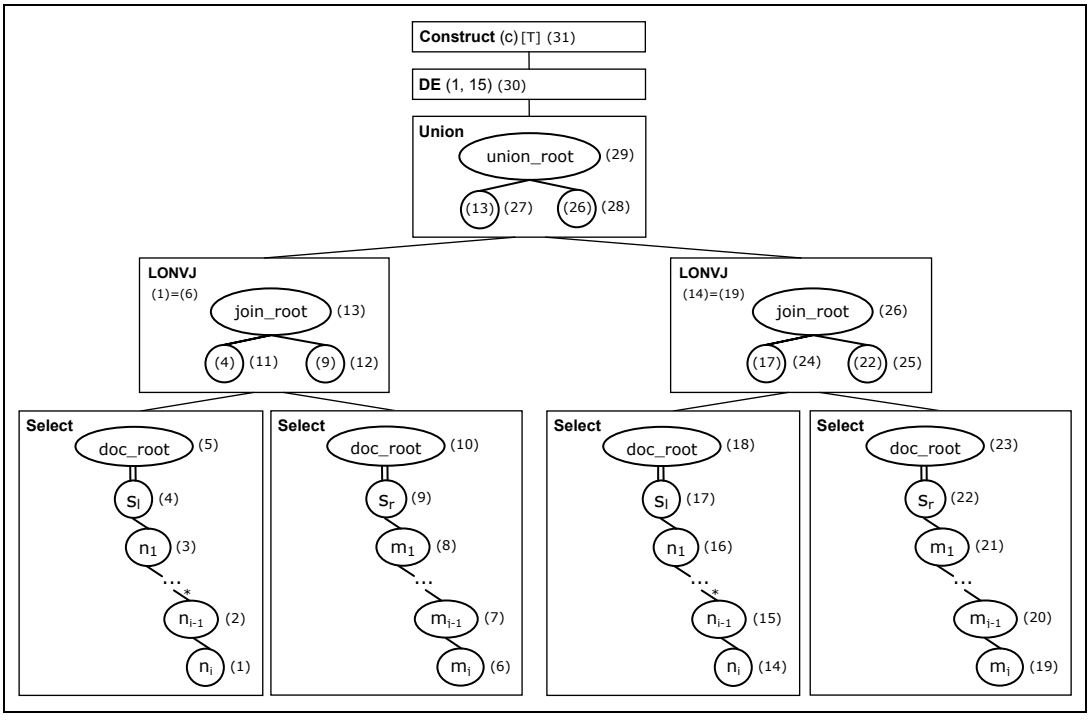


Figura 4.10 – Expressão algébrica representada por $(S_l \ddot{U}_{T,p} S_r)$

Sejam s_l e s_r as classes lógicas que representam os nós-raiz das árvores de S_l e S_r, respectivamente. Seja o predicado p dado por $[s_l/n_1/.../n_{i-1}/n_i = s_r/m_1/.../m_{j-1}/m_j]$. A expressão algébrica TLC correspondente a $(S_l \ddot{U}_{T,p} S_r)$ é apresentada na Figura 4.10.

4.3. Serviços Web

Os serviços Web [47] têm quebrado significativas barreiras para a interoperabilidade de aplicações e podem ser vistos como um novo paradigma para a extração e integração de fontes de dados (ou sistemas) heterogêneas. Alguns desafios tecnológicos relacionados ao acesso, segurança e possibilidade de descoberta automática de informações entre aplicações foram resolvidos pelos serviços Web, através de uma padronização da infraestrutura para a publicação e troca de dados. Desta forma, aplicações (ou componentes destas aplicações) podem facilmente interagir entre si independente da linguagem, plataforma ou sistema operacional em que estão executando.

A plataforma básica para a implementação de serviços Web consiste de XML e HTTP. Além disso, três outros elementos complementam esta plataforma: o *Simple Object Access Protocol* (SOAP) para conexão e comunicação sobre o protocolo HTTP; a *Web Services Description Language* (WSDL) para descrição do serviço; e o *Universal Description, Discovery and Integration* (UDDI) para implementação de serviços de diretórios. As principais operações que envolvem os Serviços Web são: publicação, descoberta, seleção, chamada e composição de serviços. Maiores detalhes sobre os serviços Web e estas operações podem ser encontradas em [47].

Dadas as vantagens apresentadas, os serviços Web têm sido fortemente sugeridos [1] como mecanismo interoperável de acesso e extração de dados de fontes distribuídas e heterogêneas (ou que estão sob sistemas distribuídos e heterogêneos).

Neste trabalho, definimos um *framework* para sistemas de mediação baseados em XML. O mediador do *framework* é um serviço Web que disponibiliza métodos para consultas sobre as VMs. Os tradutores locais também são serviços Web que disponibilizam métodos para consultas sobre as VLEs. No processamento de uma consulta sobre uma VM, o mediador gera um plano de execução que consiste de um *workflow* de processos. Estes processos disparam a execução de subconsultas sobre as VLEs (através dos serviços Web locais) e realizam operações de integração de dados.

A seguir, apresentamos mais detalhes dos *workflows* gerados neste trabalho.

4.4. *Workflows* de Execução de Consultas (Planos de Execução de Consultas)

Neste trabalho, o plano de execução de uma consulta consiste de um *workflow* de processos (*workflow* de execução da consulta). Este *workflow* é gerado pelo mediador do sistema durante o processamento da consulta. O mediador também é responsável por manter o controle do fluxo de execução do *workflow*, disparando os processos na seqüência pré-definida.

Os w*orkflows* de execução de consultas são representados neste trabalho por *grafos direcionados*, onde os nós representam processos e as arestas representam o fluxo de dados entre estes processos. Na próxima seção, definimos os tipos de processos utilizados neste *workflows*. Em seguida, apresentamos o passo-a-passo da execução de cada tipo de processo, controlado pelo mediador.

4.4.1. Processos

Os processos de um *workflow* de execução de uma consulta são implementados internamente pelo mediador e dividem-se em:

(i) ***Processos-Consulta*** (PC), responsáveis por enviar consultas aos tradutores locais (serviços Web locais) para execução sobre as VLEs; e

(ii) ***Processos-Operadores***, que realizam operações de integração de coleções XML ou filtragem de elementos destas coleções.

Conforme visto na Seção 4.2.2, os operadores de integração utilizados neste trabalho são: *Union* (U), *Outer Union* (Ü) e *Left-Outer-Nest-Value-Join* (Ĵ). Assim, os Processos-Operadores subdividem-se de acordo com a principal operação de integração que realizam, a saber:

- Processo-Union (PU); - Processo-Outer Union (PO);
- Processo-Junção Múltipla (PJM); - Processo-Filtro (PF);

Todos os processos de um *workflow* possuem atributos, parâmetros de entrada e de saída. Os valores dos atributos não são repassados no fluxo de dados entre os processos. Normalmente, estes valores são definidos na instanciação destes processos, mas podem ser atualizados a qualquer momento pelo mediador. Já os parâmetros de entrada e de saída são sempre coleções XML (conjuntos de árvores XML), representadas por documentos XML, que transitam entre os processos do *workflow*. Portanto, apenas coleções XML são repassadas no fluxo de dados entre os processos.

A seguir, a definição detalhada de cada tipo de processo.

- **Processo-Consulta (PC)**

Um *Processo-Consulta* (PC) é responsável por enviar uma consulta XQuery a um tradutor local para ser executada sobre uma VLE. Para isto, invoca o método *ConsultaVisaoXML* do tradutor local (vimos no Capítulo III a definição deste método), o qual é implementado por um serviço web, passando a consulta XQuery como parâmetro de entrada. O resultado do PC é uma coleção XML, retornada pelo tradutor local, que representa a resposta da consulta.

A consulta XQuery é um atributo do PC que, opcionalmente, pode ser gerada com uma expressão de seleção que possui parâmetros os quais deverão ser substituídos por valores escalares. Neste caso, o PC deve receber como parâmetro de entrada uma coleção XML da qual extrairá os valores escalares que substituirão os parâmetros da expressão de seleção da consulta, antes do envio para o tradutor local. Esta possibilidade de substituir parâmetros da consulta por valores escalares permite a implementação da estratégia baseada na operação de semi-junção nas operações de junção de coleções, conforme veremos no Capítulo VI.

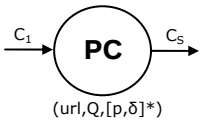
- Atributos:

1) URL do Tradutor Local (serviço Web local) que exporta a VLE;
2) Consulta XQuery que deve ser executada sobre a VLE;
3) Opcional. Conjunto de pares [p, δ/e], onde p é um parâmetro da expressão de seleção da consulta e δ/e é uma expressão de caminho sobre a coleção XML de entrada que retorna os valores escalares que substituirão o parâmetro p.

- Parâmetros:

1) Entrada: Opcional. Coleção XML de onde serão extraídos os valores escalares que substituirão os parâmetros da expressão de seleção da consulta;
2) Saída: Coleção XML com o resultado da consulta.

Caso a consulta possua uma expressão de seleção com parâmetros, todos os atributos e parâmetros do PC passam a ser obrigatórios. A URL do Tradutor Local é obtida do catálogo do mediador. A consulta será gerada pelo algoritmo de geração do *workflow* de materialização da VM, que também determina os pares [p, δ/e], quando for o caso. Na Figura 4.11 é exibida a representação gráfica de um Processo-Consulta (PC).



Onde:
url – URL do Tradutor;
Q – Consulta XQuery sobre a VLE;
[p,δ/e]* - Opcional. Conjunto de pares [p,δ/e], onde p é um parâmetro da expressão de seleção de Q e δ/e é o caminho do elemento de C_1 de onde serão extraídos os valores escalares que substituirão o parâmetro p;
C_1 – Opcional. Coleção XML para extração de valores escalares que substituirão os parâmetros da expressão de seleção de Q;
C_S – Coleção XML com o resultado de Q.

Figura 4.11 – Processo Consulta (PC)

- **Processo-Union (PU) e Processo-Outer Union (PO)**

Processo-Union (PU) e *Processo-Outer Union (PO)* realizam a integração de duas ou mais coleções XML de acordo com as definições dos operadores *Union* (U) e *Outer Union* ($Ü_{T,p}$), respectivamente.

- Atributos do Processo PU: Não possui;

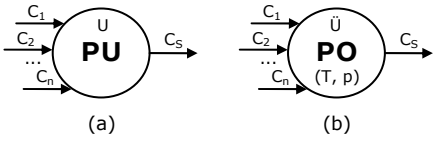
- Atributos do Processos PO:

1) Tipo XML T que servirá de base para a operação de *Outer Union* ($Ü_{T,p}$).
2) Predicado p com o critério de junção (duas operações de *Left-Outer-Nest-Value-Join*, em sentidos contrários) das coleções;

- Parâmetros:

1) Entrada: Lista de coleções XML;
2) Saída: Coleção XML com o resultado da integração das coleções, segundo o operador de integração utilizado pelo processo.

Nas Figuras 4.12(a) e 4.12(b) são exibidas, respectivamente, as representações gráficas de um Processo-Union (PU) e de um Processo-Outer Union (PO).



Onde:
C_1 , C_2 ... C_n – Lista de coleções XML de entrada;
T – Tipo XML utilizado como base;
p – predicado com critério de junção das coleções de entrada;
C_S – Coleção XML com o resultado das operações de integração;
U, Ü – Operadores de integração.

Figura 4.12 – (a) Processo Union; (b) Processo Outer Union;

- **Processo-Junção Múltipla (PJM)**

Um *Processo-Junção Múltipla (PJM)* realiza a junção de uma coleção XML C_B com uma ou mais coleções XML, utilizando o operador *Left-Outer-Nest-Value-Join* ($Ĵ_{T,p}$). A coleção C_B é chamada de *Coleção Base* do PJM.

Um conjunto de pares [ℓ, T], onde T é um tipo XML e ℓ é uma ligação entre C_B (ou uma coleção aninhada de C_B) e uma coleção XML de entrada Ci, determina como cada operação de junção ($Ĵ_{T,p}$) deve ser realizada.

Assim, para cada par [ℓ, T] , onde

- ℓ é ligação entre C_B e C_i , ou seja, $\ell : C_B\{a_1, ..., a_n\} \longrightarrow C_i\{b_1, ..., b_n\}$ ou
- ℓ é ligação entre uma coleção aninhada de C_B e a coleção C_i , ou seja,

$\ell : C_B(\delta)\{a_1, ..., a_n\} \longrightarrow C_i\{b_1, ..., b_n\}$

A operação de junção realizada é ($C_B \hat{J}_{T,a1=b1\wedge...\wedge an=bn} C_i$).

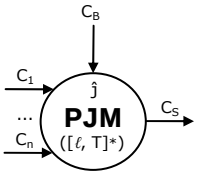
- Atributos:

1) Conjunto de pares [ℓ, T].

- Parâmetros:

1) Entrada: Coleção Base C_B e um conjunto de coleções XML.
2) Saída: Coleção XML com o resultado da junção de C_B com as demais coleções XML de entrada.

Na Figura 4.13 é exibida a representação gráfica de um Processo-Junção Múltipla (PJM).



Onde:
C_B – Coleção Base, a qual será integrada por junção às demais coleções de entrada;
C_1 ... C_n – Conjunto de coleções XML que sofrerão junção, uma a uma, com C_B;
[ℓ, T]* – Conjunto de pares [ℓ, T], que determina como cada operação de junção ($\hat{J}_{T,\delta}$) deve ser realizada entre C_B e as demais coleções XML de entrada C_1 ... C_n;
C_S – Coleção XML com o resultado de PJM
$\hat{J}$ – Operador de integração Left-Outer-Nest-Value-Join

Figura 4.13 – Processo Junção Múltipla (PJM)

- **Processo-Filtro (PF)**

Um *Processo-Filtro (PF)* é um processo que recebe uma coleção XML de entrada e filtra os seus elementos primários conforme uma expressão de seleção Exp. O resultado é uma coleção XML formada por elementos oriundos da coleção XML de entrada que satisfazem Exp.

- Atributos:

1) Expressão de seleção Exp;

- Parâmetros:

1) Entrada: Coleção XML;
2) Saída: Coleção XML de elementos primários da coleção de entrada que satisfazem Exp.

Na Figura 4.14 é exibida a representação gráfica de um Processo-Filtro (PF).

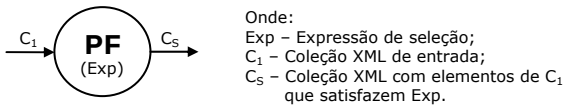


Figura 4.14 – Processo Filtro (PF)

4.4.2. Execução de Processos

O objetivo desta seção é mostrar o passo-a-passo da execução dos tipos de processos de um *workflow* de execução de uma consulta. A execução destes *workflows* é sempre iniciada pela execução do nó-raiz dos grafos que os representam. De acordo com as regras de execução apresentadas a seguir, os demais nós do grafo serão também executados numa espécie de execução em cascata.

Neste trabalho, o termo "execução de um nó do grafo" não consiste simplesmente em executar o processo representado por aquele nó, pois muitas vezes ele depende de resultados obtidos da execução de outros nós. Assim, "executar um nó do grafo" consiste em avaliar as dependências existentes, executá-las primeiramente e, por fim, executar o processo representado pelo nó em questão.

No que se segue, para se ter uma noção explícita da seqüência de execução dos processos do grafo de resolução, inclusive das execuções em paralelo, serão exibidos diagramas de bloco como ilustração. Considere:

- *Nó-{Processo-X}*: um nó que representa determinado *Processo X*.
- *Nó-Operador*: qualquer nó que represente um Processo-Operador.
- *Nó-Fornecedor*: qualquer nó N_2 que esteja ligado a outro nó N_1 e forneça o seu resultado como entrada do processo representado por N_1.

A execução de um Nó-Processo-Consulta é trivial e não demanda a definição de uma regra de execução. Nestes casos, a execução do nó resume-se à execução do Processo-Consulta.

Regra de Execução 1: Execução de um Nó-Processo-*Outer-Union* (PO)

Um nó que representa um Processo *Outer Union* (PO) deve ser executado conforme os seguintes passos:

1) Executar em paralelo cada um dos seus Nós-Fornecedores;
2) Executar o Processo *Outer Union* (PO), tomando como entrada os resultados do passo 1.

Exemplo: Considere o grafo exibido na Figura 4.15(a). O diagrama de blocos de execução do Nó-Processo-*Outer-Union* PO, de acordo com a Regra de Execução 1, é exibido na Figura 4.15(b).

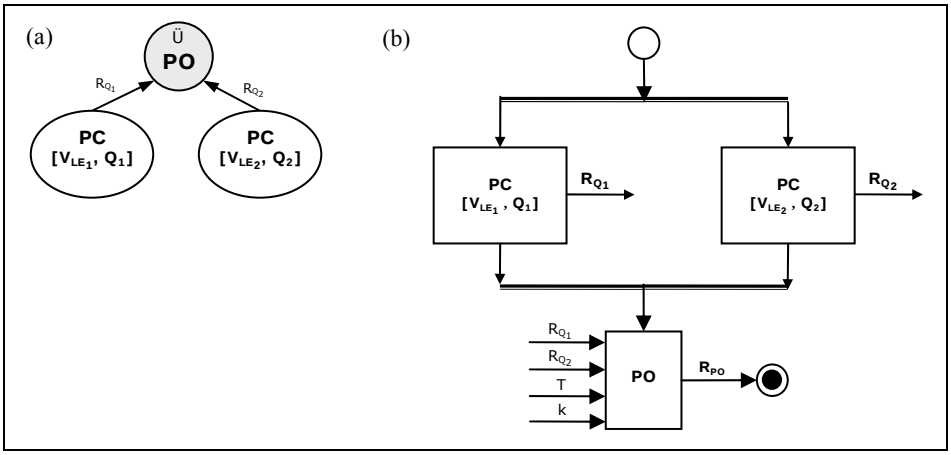


Figura 4.15 – (a) Nó-Processo-*Outer-Union* PO ; (b) Diagrama de Blocos de Execução de PO

Regra de Execução 2: Execução de um Nó-Processo-*Union* (PU)

Um nó que representa um Processo *Union* (PU) deve ser executado conforme os seguintes passos:

1) Executar em paralelo cada um dos seus Nós-Fornecedores;
2) Executar o Processo *Union* (PU), tomando como entrada os resultados do passo 1.

Exemplo: Considere o grafo exibido na Figura 4.16(a). O diagrama de blocos de execução do Nó-Processo-*Union* PU, de acordo com a Regra de Execução 2, é exibido na Figura 4.16(b). Note que D_1 é o diagrama de blocos de execução do Nó-Processo-*Outer-Union* PO.

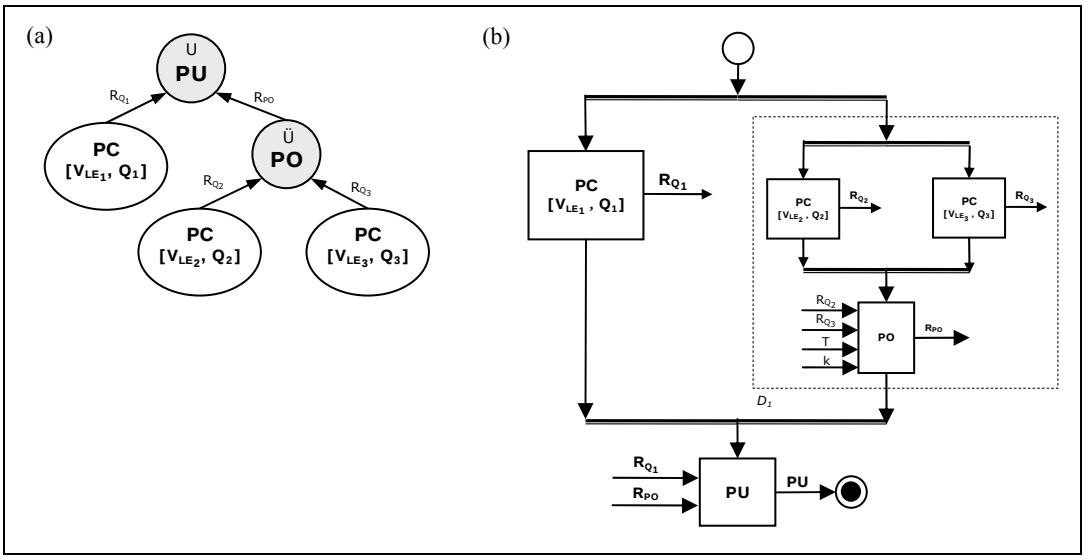


Figura 4.16 – (a) Nó-Processo-*Union* PU ; (b) Diagrama de Blocos de Execução de PU

Regra de Execução 3: Execução de um Nó-Processo-Filtro (PF)

É importante observar que um nó que representa um Processo Filtro (PF) possui sempre um único Nó-Fornecedor, que é um Nó-Processo-*Outer-Union* (PO). Assim, a execução de um Nó-Processo-Filtro PF deve ser realizada conforme os seguintes passos:

1) Executar o Nó-Fornecedor (Nó-Processo-*Outer-Union*);
2) Executar o Processo-Filtro, tomando como entrada o resultado do passo 1.

Por ser trivial, não apresentaremos um exemplo de execução de um Nó-Processo-Filtro.

Regra de Execução 4: Execução de um Nó-Processo-Junção-Múltipla (PJM)

A coleção-base de um Processo Junção Múltipla (vide Seção 4.4.1) é fornecida por um nó o qual chamaremos de *Nó-Base*.

Um nó que representa um Processo Junção Múltipla (PJM) deve ser executado conforme os seguintes passos:

1) Executar em paralelo os Nós-Fornecedores à esquerda[1] do Nó-Base;
2) Executar o Nó-Base, tomando como entrada os resultados do passo 1;
3) Executar em paralelo os demais Nós-Fornecedores N', com as seguintes condições:

   3.1) Se N' é um Nó-Processo-Consulta $PC[V_{LE_i}]$, tomar como entrada o resultado do passo 2;

   3.2) Se N' é um Nó-Operador, então para qualquer caminho δ=N'/.../N" do grafo, onde N" é um Nó-Processo-Consulta $PC[V_{LE_i}]$, tomar como entrada na execução de N" o resultado do passo 2. Em outras palavras, os Nós-Processo-Consulta N" que estiverem "abaixo" de N' deverão tomar como entrada o resultado do passo 2;
4) Executar o Processo Junção Múltipla (PJM), tomando como entrada o resultado do passo 2 (coleção base), os resultados do passo 1 e os resultados do passo 3.

[1] Neste trabalho, qualquer Nó-Fornecedor possui um atributo inteiro que identifica sua ordem como fornecedor de coleções XML para um "nó-pai". Considerando o valor deste atributo, os Nós-Fornecedores estão visualmente ordenados de forma crescente da esquerda para a direita, em relação ao seu "nó-pai". Assim, os nós que estão à esquerda ou à direita de um determinado nó possuem, respectivamente, valores menores ou maiores para estes atributos.

Exemplo: Considere o grafo exibido na Figura 4.17(a). O diagrama de blocos de execução do Nó-Processo-Junção-Múltipla PJM, de acordo com a Regra de Execução 3, é exibido na Figura 4.17(b). O Nó-Processo-Consulta PC[V_{LE1}, Q_1] é o Nó-Base. Note que, conforme o passo 3.2 da regra, os Nós-Processo-Consulta PC[V_{LE4}, Q_4] e PC[V_{LE5}, Q_5] tomam como entrada o resultado da execução do Nó-Base PC[V_{LE1}, Q_1]. Observe ainda que D_1 é o diagrama de blocos de execução do Nó-Processo-*Outer-Union* PO.

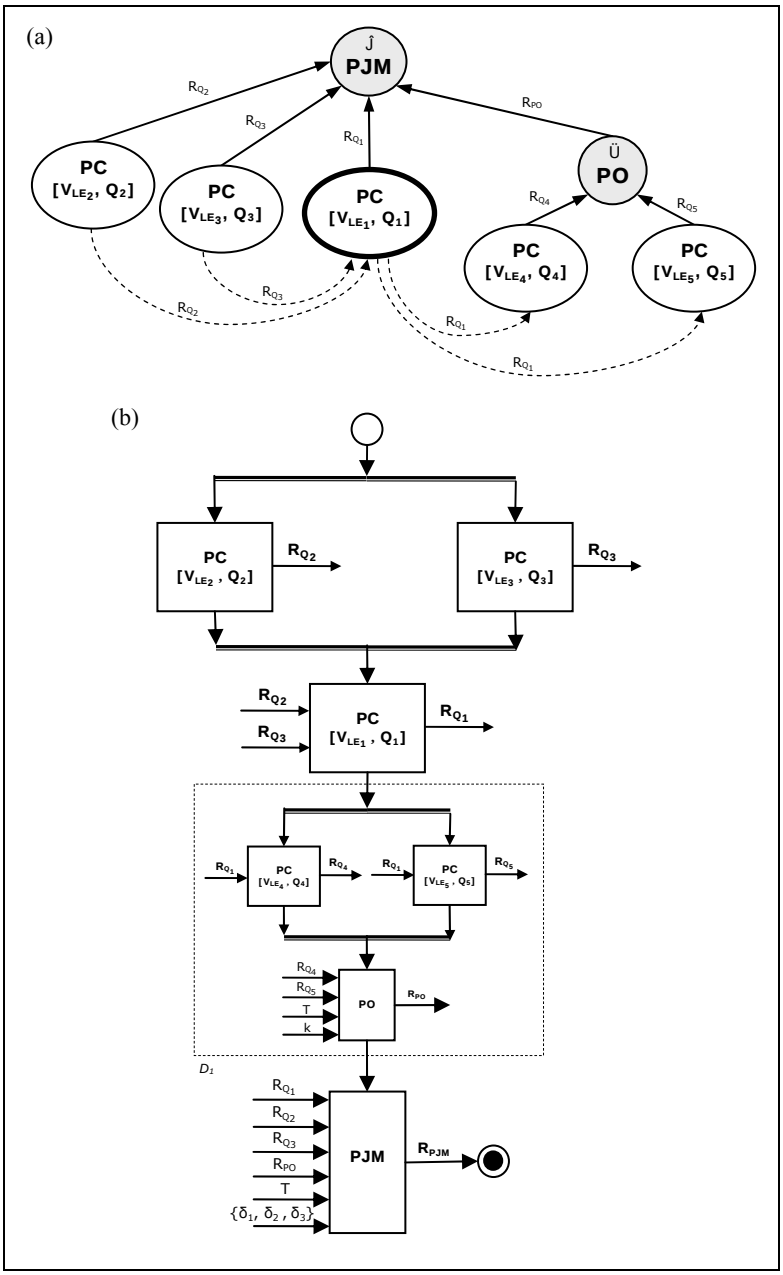


Figura 4.17 – (a) Nó-Processo-Junção-Múltipla PJM ; (b) Diagrama de Blocos de Execução de PJM

Capítulo V

Especificação e Geração de Visões de Mediação

Neste capítulo é apresentada a abordagem para especificação de uma Visão de Mediação diretamente sobre fontes de dados heterogêneas (VM heterogênea) e o processo de transformação de uma VM heterogênea em uma VM equivalente definida em termos de visões XML locais, as quais são fragmentos homogêneos da VM (VM homogênea).

5.1. Introdução

O advento do XML como formato padrão para troca de dados na Web aumentou a demanda pela publicação, no formato XML, de dados que se encontram armazenados nos mais diversos modelos e formatos. Uma maneira simples de publicar dados no formato XML é gerar visões XML sobre aqueles dados [37][42]. As visões XML podem ser virtuais ou materializadas, seguindo o mesmo conceito das abordagens apresentadas no Capítulo II. A publicação de dados através de visões XML é tratada em vários trabalhos [18][38][44].

Além do problema de publicação de dados no formato XML, cada vez mais aplicações necessitam integrar dados de fontes heterogêneas e distribuídas através da Web e este problema também vem sendo estudado por diversos pesquisadores.

As visões XML são também apresentadas como proposta para integração de dados na Web, especialmente em sistemas baseados em mediação [6][22][27], cujos principais componentes são: o esquema da visão de mediação (ou visão XML de integração de dados); os esquemas das fontes de dados e os mapeamentos que especificam as correspondências entre a visão de mediação e as fontes de dados. Estes sistemas seguem, portanto, uma arquitetura de dois níveis de esquemas.

Os dois principais problemas relacionados à utilização de visões de mediação para a integração de dados são: (i) especificar a visão de mediação, sendo necessário identificar quais fontes deverão prover os dados e definir um formalismo para especificar o mapeamento entre o esquema da visão e os esquemas das fontes de dados; e (ii) realizar o processamento de consultas sobre estas visões, ou seja, definir como os

mapeamentos podem ser utilizados para responder corretamente e eficientemente consultas sobre a visão de mediação.

Neste capítulo, apresentamos a abordagem para especificação de uma VM diretamente sobre fontes de dados heterogêneas (VM heterogênea) e o processo de transformação de uma VM heterogênea em uma VM equivalente definida em termos de visões XML locais, as quais são fragmentos homogêneos da VM (VM homogênea).

A principal contribuição da abordagem proposta são as ***Assertivas de Correspondência***, um formalismo simples, mas expressivo, que permite especificar os mapeamentos entre o esquema da VM heterogênea e os esquemas das múltiplas fontes de dados, suportando inclusive esquemas locais com estruturas aninhadas, como bases XML nativas. Durante o processo de transformação de uma VM heterogênea em uma VM homogênea, as visões XML locais são automaticamente geradas a partir destas assertivas.

5.2. Assertivas de Correspondência

Um passo fundamental na especificação de uma VM heterogênea é definir como os seus elementos são obtidos a partir das diversas fontes de dados, ou seja, definir o mapeamento entre a VM heterogênea e as fontes de dados.

Nesta seção, são apresentadas as ***Assertivas de Correspondência*** (ACs), um formalismo, estendido de [29], que permite ao projetista especificar como os elementos da VM são obtidos a partir das múltiplas fontes de dados. As ACs são definidas apenas entre coleções XML (uma VM é uma extensão do conceito de coleção XML). Desta forma, propõe-se um mecanismo de mapeamento flexível, onde uma coleção XML local pode representar uma tabela relacional, uma tabela de objetos, um fragmento de documento XML, etc. No caso de uma tabela relacional, por exemplo, atributos e restrições referenciais (*foreign keys*) são mapeados em elementos e ligações da coleção XML correspondente, respectivamente.

As ACs estão divididas em dois tipos: (i) *Assertivas de Correspondência Global ou de Coleção (ACG)*, que permitem especificar como os estados de duas coleções estão correlacionados; e (ii) *Assertivas de Correspondência de Caminho (ACC)*, que permitem especificar o mapeamento entre os esquemas de duas coleções.

Todas as ACs são especificadas pelo projetista da VM heterogênea de forma manual. A seguir, são apresentadas as definições formais das ACs e outras definições relacionadas, utilizadas no decorrer deste trabalho.

Definição 5.1: Assertiva de Correspondência Global ou de Coleção (ACG)

Sejam **C** e **C'** coleções XML cujas chaves são dadas por k e k', respectivamente. Uma *Assertiva de Correspondência Global ou de Coleção* (ACG) é uma expressão da forma:

- [**C**] ≡ [**C'**], onde:
  - se $e ∈ **$C** então ∃ $e' ∈ **$C'** tal que $e/k = $e'/k' e
  - se $e' ∈ **$C'** então ∃ $e ∈ **$C** tal que $e'/k' = $e/k;
- [**C**] ⊃ [**C'**], onde:
  - se $e' ∈ **$C'** então ∃ $e ∈ **$C** tal que $e'/k' = $e/k;
- [**C**[exp]] ≡ [**C'**], onde:
  - exp é uma expressão de seleção sobre **C**;
  - se $e ∈ **$C**[exp] então ∃ $e' ∈ **$C'** tal que $e/k = $e'/k' e
  - se $e' ∈ **$C'** então ∃ $e ∈ **$C**[exp] tal que $e'/k' = $e/k.
- [**C**[exp] ∩ **C'**] = [Ø], onde:
  - exp é uma expressão de seleção sobre **C**;
  - se $e ∈ **$C**[exp] então ¬∃ $e' ∈ **$C'** tal que $e/k = $e'/k' e
  - se $e' ∈ **$C'** então ¬∃ $e ∈ **$C**[exp] tal que $e'/k' = $e/k. □

Definição 5.2: Assertiva de Correspondência de Caminho (ACC)

Sejam T e T' tipos XML complexos restritos. Suponha que e é um elemento/atributo de T e que δ é um caminho de T', o qual pode ser não definido (δ=∅) ou nulo (δ=NULL). Diz-se que e é compatível com δ sse: (i) δ=∅; ou (ii) δ=NULL e T_e (tipo de e) é complexo; ou (iii) T_e e T_δ (tipo de δ) são ambos simples ou complexos e possuem a mesma cardinalidade.

Uma *Assertiva de Correspondência de Caminho* (ACC) é uma expressão da forma:

- [T/e] ≡ [T'/δ] , tal que e é compatível com δ; ou
- [T/e] ⊃ [T'/δ] , tal que e é compatível com δ; ou
- [T/e] ≡ [T'/δ/ℓ] , tal que e é compatível com δ e ℓ é uma ligação da coleção em T'/δ; ou
- [T/e] ⊃ [T'/δ/ℓ] , tal que e é compatível com δ e ℓ é uma ligação da coleção em T'/δ. □

Definição 5.3: Assertivas de Junção

As ACCs da forma [T/e] ≡ [T'/δ/ℓ] ou [T/e] ⊃ [T'/δ/ℓ] são denominadas *Assertivas de Junção*. □

Definição 5.4: Seja $\mathcal{A}$ um conjunto de assertivas de correspondência de caminho. Dizemos que $\mathcal{A}$ *especifica completamente* T *em termos de* T' sse:

1) Para cada elemento e de T, existe:
   - uma única ACC para e em $\mathcal{A}$ da forma [T/e] ≡ [T'/δ] ou [T/e] ⊃ [T'/δ]; e
   - uma única ou nenhuma ACC para e em $\mathcal{A}$ da forma [T/e] ≡ [T'/δ/ℓ] ou [T/e] ⊃ [T'/δ/ℓ]
2) Para cada assertiva em $\mathcal{A}$ da forma [T/e] ≡ [T'/δ] ou [T/e] ⊃ [T'/δ], onde e tem tipo complexo T_e, então $\mathcal{A}$ especifica completamente T_e em termos de T_δ (tipo de δ).
3) Para cada assertiva em $\mathcal{A}$ da forma [T/e] ≡ [T'/δ/ℓ] ou [T/e] ⊃ [T'/δ/ℓ], onde e tem tipo complexo T_e e T_ℓ é o tipo do elemento complexo referenciado por ℓ, então:
   - $\mathcal{A}$ especifica completamente T_e em termos de T_δ; e
   - $\mathcal{A}$ especifica completamente T_e em termos de T_ℓ. □

Definição 5.5: Seja $\mathcal{A}$ um conjunto de assertivas de correspondência de caminho tal que $\mathcal{A}$ especifica completamente T em termos de T'.

1) Sejam $\mathbf{S}_1$ e $\mathbf{S}_2$ conjuntos de elementos XML instâncias dos tipos XML simples T_1 e T_2, respectivamente. Dizemos que $\mathbf{S}_1 \equiv_{\mathcal{A}} \mathbf{S}_2$ sse $\mathbf{\$t_1} \in \mathbf{S}_1$ sse existe $\mathbf{\$t_2} \in \mathbf{S}_2$ tal que $\mathbf{\$t_1}$ / text() = $\mathbf{\$t_2}$ / text().
2) Sejam $\mathbf{S}_1$ e $\mathbf{S}_2$ conjuntos de elementos XML instâncias dos tipos XML complexos T_1 e T_2, respectiv. Dizemos que $\mathbf{S}_1 \equiv_{\mathcal{A}} \mathbf{S}_2$ sse $\mathbf{\$t_1} \in \mathbf{S}_1$ sse existe $\mathbf{\$t_2} \in \mathbf{S}_2$ tal que $\mathbf{\$t_1} \equiv_{\mathcal{A}} \mathbf{\$t_2}$.
3) Sejam $\mathbf{S}_1$ e $\mathbf{S}_2$ conjuntos de elementos XML instâncias dos tipos XML simples T_1 e T_2, respectivamente. Dizemos que $\mathbf{S}_1 \supset_{\mathcal{A}} \mathbf{S}_2$ sse se existe $\mathbf{\$t_2} \in \mathbf{S}_2$, então ∃ $\mathbf{\$t_1} \in \mathbf{S}_1$, tal que $\mathbf{\$t_1}$ / text() = $\mathbf{\$t_2}$ / text().
4) Sejam $\mathbf{S}_1$ e $\mathbf{S}_2$ conjuntos de elementos XML instâncias dos tipos XML complexos T_1 e T_2, respectivamente. Dizemos que $\mathbf{S}_1 \supset_{\mathcal{A}} \mathbf{S}_2$ sse se existe $\mathbf{\$t_2} \in \mathbf{S}_2$, então ∃ $\mathbf{\$t_1} \in \mathbf{S}_1$, tal que $\mathbf{\$t_1} \equiv_{\mathcal{A}} \mathbf{\$t_2}$.
5) Seja **$t** uma instância de T e seja **$t'** uma instância de T'. Dizemos que **$t** $\equiv_{\mathcal{A}}$ **$t'** sse, para cada elemento/atributo e de T:
   - se [T/e] ≡ [T'/δ] é uma ACC para e em $\mathcal{A}$, então **$t**/e $\equiv_{\mathcal{A}}$ **$t'**/δ, onde δ≠∅ e **$t**/e ≠∅;
   - se [T/e] ⊃ [T'/δ] é uma ACC para e em $\mathcal{A}$, então **$t**/e $\supset_{\mathcal{A}}$ **$t'**/δ, onde **$t**/e ≠ ∅;
   - se [T/e] ≡ [T'/δ/ℓ] é uma ACC para e em $\mathcal{A}$, então **$t**/e $\equiv_{\mathcal{A}}$ **$t'**/δ e **$t**/e $\equiv_{\mathcal{A}}$ **$t'**/δ/$\ell$, onde δ≠∅ e **$t**/e ≠∅;

◦ se [T/e] ⊃ [T'/δ/ℓ] é uma ACC para e em $\mathcal{A}$, então **$t**/e ⊃$_{\mathcal{A}}$ **$t'**/δ e **$t**/e ⊃$_{\mathcal{A}}$ **$t'**/δ/ℓ, onde **$t**/e ≠∅.

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

□

Definição 5.6: Seja $\mathcal{A}$ um conjunto de assertivas de correspondência de caminho que especifica completamente T em termos de T_1 e … e T_n. Denotamos por $\mathcal{A}_{[T\&Ti]}$ o conjunto de assertivas de $\mathcal{A}$ que especificam completamente T em termos de T_i, 1≤i≤n. □

5.3. Especificação de uma Visão de Mediação Heterogênea

Nesta seção, apresentamos a abordagem para especificação de uma VM diretamente sobre fontes de dados heterogêneas e distribuídas (VM heterogênea), utilizando como formalismo de mapeamento as Assertivas de Correspondência.

Inicialmente, vejamos a definição de uma VM heterogênea.

Definição 5.7: Visão de Mediação Heterogênea (VM Heterogênea)

Uma *Visão de Mediação Heterogênea* é definida por uma tripla **V' = <S**, **L**, Υ>, onde:

◦ **S** = <e,T> é o ***esquema de mediação*,** onde e é o nome do elemento primário da visão e T é o tipo XML de e;

◦ **L** = <{L_1, ..., L_n}, R'> é o ***esquema base***, formado pela união dos esquemas locais e restrições entre estes esquemas. No esquema base, L_i , i=1,...,n , é o esquema de uma fonte de dados local e R' é um conjunto formado por restrições de interseção e ligações (vide Capítulo IV) entre coleções de fontes locais distintas.

◦ Υ = <ψ', $\mathcal{A}$ > são os ***mapeamentos de* V'**, onde:

- ψ' = $\bigcup_{i=1}^{n} \psi'_i$, onde ψ'_i é o conjunto de Assertivas de Correspondência Global ou de Coleção (ACG) entre **V'** e uma coleção **C_i** de um esquema local L_i , chamada de *coleção-base*. O conjunto ψ'_i possui, no mínimo, uma ACG do tipo **[V']** ≡ **[C_i]** ou **[V']** ⊃ **[C_i]** ; e zero ou mais ACGs do tipo **[V'**[exp]] ≡ **[C_i]**.

- $\mathcal{A}$ = $\bigcup_{i=1}^{n} \mathcal{A}_i$, onde $\mathcal{A}_i$ é o conjunto de Assertivas de Correspondência de Caminho (ACC) que *especificam completamente* T *em termos de* T_{Ci} (o tipo de **C_i**). Desta forma, para toda instância **$t** de T existe uma instância **$c** de T_{Ci} tal que **$t** *é semanticamente equivalente a* **$c** *conforme especificado por* $\mathcal{A}_i$ (**$t** ≡$_{\mathcal{A}_i}$ **$c**). □

Assim, a especificação de uma VM heterogênea é composta por três atividades:

(i) **Definição do esquema de mediação** (**S**);

(ii) **Definição do esquema base** (**L**);

(iii) **Definição dos mapeamentos da VM heterogênea** (γ);

Na atividade *Definição do esquema de mediação*, o projetista define o nome e o tipo XML do elemento primário da VM. Em [45] é proposta uma ferramenta gráfica que auxilia na geração do esquema de uma visão e que é utilizada neste trabalho.

Em seguida, na atividade *Definição do esquema base*, o projetista informa quais as fontes que deverão prover dados para compor a VM e, opcionalmente, define as restrições de interseção e as ligações entre coleções de fontes de dados distintas.

É importante observar que a possibilidade de se definir restrições de interseção e ligações entre coleções de fontes distintas é um diferencial deste trabalho. Em relação às restrições de interseção, veremos adiante que elas serão utilizadas para gerar novas restrições de interseção entre as visões XML locais. Desta forma, no processamento de uma consulta, dados oriundos de visões XML locais disjuntas serão integrados por operadores de menor custo computacional, favorecendo o desempenho do algoritmo de processamento. Em relação às ligações entre coleções de fontes distintas, veremos a seguir que, através destas ligações, durante a definição das assertivas, é possível navegar entre elementos/atributos de coleções de bases distintas e definir VMs com uma maior riqueza de informações.

Finalmente, na atividade *Definição dos mapeamentos da VM heterogênea*, o projetista: (i) define as ACGs entre a VM e as coleções-base; e (ii) realiza o *matching* do tipo da VM com os tipos das coleções-base.

Uma *coleção-base* é uma coleção que possui um mapeamento 1-1 entre o seu elemento primário e o elemento primário da VM, ou seja, o seu elemento primário representa a mesma entidade do mundo real representada pelo elemento primário da VM. As coleções-base são identificadas em cada fonte de dados pelo projetista. Por exemplo, na VM Pacientes do Capítulo III, as tabelas BD_1:Pessoa e BD_2:Paciente foram escolhidas pelo projetista como coleções-base. Já as tabelas BD_3:Patologia e BD_4:Medico não podem ser ccleções-base de Pacientes, pois os seus elementos primários (<Patologia> e <Medico>, respectivamente) não representam a mesma entidade do mundo real representada pelo elemento primário <Paciente> da VM.

No processo de *matching*, o projetista define o conjunto de ACCs que especifica completamente o tipo da VM em termos do tipo de cada coleção-base. A ferramenta proposta em [45] apresenta uma interface gráfica na qual é possível definir o tipo da visão em termos de uma única base de dados. Essa ferramenta é adaptada neste trabalho para permitir a definição de ACGs e a definição do tipo da visão em termos de mais de uma fonte de dados, utilizando inclusive as ligações entre coleções de fontes distintas.

Os passos para a definição das ACs utilizando a ferramenta são os seguintes:

1) O projetista escolhe uma coleção-base;
2) A ferramenta solicita a definição das ACGs da VM em relação àquela coleção-base;
3) O projetista define as ACGs;
4) A ferramenta exibe, em formato de árvore de diretórios, o tipo da VM e o tipo da coleção-base vinculado aos tipos das coleções que possuem ligações com a coleção-base. A partir da coleção-base, o projetista pode navegar para atributos das coleções que possuem ligação com a coleção-base;
5) O projetista define graficamente as ACCs conectando os elementos/atributos do tipo da VM com os elementos/atributos da coleção-base ou das coleções ligadas à coleção-base. De forma recursiva, são definidas também as ACCs para os subelementos do tipo da VM.
6) O processo é repetido para todas as coleções-base.

Alguns mecanismos (semi)automáticos como, por exemplo, abordagens baseadas no uso de ontologias [11][28], podem ser utilizados para analisar as bases de dados e sugerir as assertivas entre o esquema da VM e os esquemas das bases de dados. Entretanto, está fora do escopo deste trabalho investigar alternativas para a obtenção das assertivas sem a intervenção humana, ou seja, a intervenção do projetista.

5.4. Geração de uma Visão de Mediação Homogênea

Neste capítulo, apresentamos o processo de transformação da especificação de uma VM heterogênea em uma VM equivalente definida em termos de visões XML locais, as quais são fragmentos homogêneos da VM (VM homogênea). Este processo de transformação é também chamado de *geração de uma VM homogênea*.

Inicialmente, vejamos a definição de uma VM homogênea.

Definição 5.8: Visão de Mediação Homogênea (VM Homogênea)

Uma *Visão de Mediação Homogênea* é definida por uma tripla $\mathbf{V} = \langle \mathbf{S}, \mathbf{E}, \mathcal{M} \rangle$, onde:

◦ $\mathbf{S} = \langle e, T \rangle$ é o ***esquema de mediação,*** onde e é o nome do elemento primário da visão e T é o tipo XML de e;

◦ $\mathbf{E} = \langle \{V_{LE1}, ..., V_{LEn}\}, R \rangle$ é o ***esquema exportado***, formado pela união das VLEs e restrições entre estas VLEs. No esquema exportado, V_{LEi} , $i=1,...,n$, é uma VLE definida em termos de uma única fonte local em L e R é um conjunto formado por restrições de interseção e ligações entre as VLEs;

◦ $\mathcal{M} = \langle \psi, \Phi \rangle$ são os ***mapeamentos de*** $\mathbf{V}$, onde:

- $\psi = \bigcup_{i=1}^{n} \psi_i$, onde ψ_i é o conjunto de Assertivas de Correspondência Global ou de Coleção (ACG) entre $\mathbf{V}$ e $\mathbf{V_{LEi}}$. O conjunto ψ_i possui, no mínimo, uma ACG do tipo $[\mathbf{V}] \equiv [\mathbf{V_{LEi}}]$ ou $[\mathbf{V}] \supset [\mathbf{V_{LEi}}]$; e zero ou mais ACGs do tipo $[\mathbf{V}[exp]] \equiv [\mathbf{V_{LEi}}]$.

- Φ é uma expressão algébrica que representa o plano de reconstrução de $\mathbf{V}$ a partir de operações de união e junção das VLEs do esquema exportado E. □

Assim, a geração de uma VM homogênea consiste de três atividades:

(i) **Geração do esquema de mediação** ($\mathbf{S}$);

(ii) **Geração do esquema exportado** ($\mathbf{E}$);

(iii) **Geração dos mapeamentos da VM homogênea** ($\mathcal{M}$);

O esquema de mediação de uma VM homogênea é o mesmo da VM heterogênea a partir da qual a VM homogênea foi gerada. Assim, a *Geração do esquema de mediação* consiste apenas da atribuição do esquema da VM heterogênea como esquema da VM homogênea.

Na *Geração do esquema exportado*, a partir das assertivas definidas entre a VM heterogênea e as fontes de dados (conjunto de mapeamentos $\mathcal{Y}$) e das restrições e ligações entre coleções das fontes de dados (conjunto R'), são geradas as VLEs e inferidas as restrições de interseção e ligações entre estas VLEs.

Por fim, na *Geração dos mapeamentos da VM homogênea* (também chamados de *mapeamentos de mediação*) são geradas as assertivas ACGs do conjunto ψ e a expressão algébrica Φ de reconstrução da VM a partir das VLEs. Conforme veremos no Capítulo VI, esta expressão é utilizada como base para a geração do plano de execução de uma consulta sobre a VM homogênea.

De fato, as três atividades do processo de geração de um VM homogênea são executadas simultaneamente pelo algoritmo **GeraVMHomogenea**, apresentado na Seção 5.6. A seguir, apresentamos maiores detalhes relacionados às atividades de geração do esquema exportado (E) e dos mapeamentos de mediação ($\mathcal{M}$) de uma VM homogênea.

5.4.1. Geração do Esquema Exportado

Na integração de dados, um problema crítico a ser tratado é a heterogeneidade estrutural entre os esquemas das bases de dados [43]. Neste trabalho, este problema é minimizado através da geração, sobre o esquema de cada base, de visões XML que são fragmentos homogêneos (verticais, horizontais ou híbridos) da visão de mediação. Estas visões são geradas automaticamente a partir da especificação de uma VM heterogênea e são denominadas ***Visões Locais Exportadas (VLEs)***.

Uma VLE é definida por uma tripla $\mathbf{V_{LE} = <S_{VLE}, L_{VLE}, \mathcal{Y}_{VLE}>}$, onde S_{VLE} é o esquema da visão, L_{VLE} é o esquema de uma fonte de dados e $\mathcal{Y}_{VLE}$ é o conjunto de assertivas que especificam o mapeamento entre S_{VLE} e L_{VLE}. O esquema da VLE é definido pelo par $<m, T_{VLE}>$, onde m é o elemento primário da VLE e T_{VLE} é o tipo XML de m, também chamado de tipo da VLE.

O fato das VLEs representarem fragmentos homogêneos da VM simplifica o processamento de consultas, pois a reescrita de uma consulta sobre a VM em subconsultas sobre as VLEs é feita de forma trivial, sem a necessidade da utilização de mapeamentos de esquema em tempo de execução, uma vez que os elementos selecionados na consulta possuem a mesma estrutura tanto na VM quanto nas VLEs. Além disso, a composição da resposta da consulta pode ser realizada apenas por operações de união e junção dos resultados das subconsultas sobre as VLEs, o que também simplifica o processamento.

A geração do esquema exportado (**E**) está divida em dois passos:

(i) Geração das VLEs Primárias;

(ii) Geração das VLEs Secundárias.

- **Geração das VLEs Primárias**

Na *Geração das VLEs Primárias* são geradas VLEs que exportam dados das coleções-base e de coleções referenciadas, dentro da mesma base de dados, pelas coleções-base. O tipo de uma VLE Primária é gerado com base nos elementos da VM heterogênea que possuem assertivas de correspondência não-vazia (∅) com a coleção-

base. Neste passo, são também geradas as Assertivas de Correspondência de Global ou de Coleção (ACGs) entre a VM homogênea e as VLEs Primárias (conjunto ψ) e, a partir das restrições de interseção entre as coleções-base, são inferidas as restrições de interseção entre as VLEs Primárias que exportam dados destas coleções-base.

- **Geração das VLEs Secundárias**

Na *Geração das VLEs Secundárias* são geradas VLEs que exportam dados que complementam as VLEs Primárias, a partir de bases de dados que não possuem coleções-base. Este passo é realizado apenas se existirem Assertivas de Junção (vide Definição 5.3) no conjunto de assertivas da VM heterogênea, ou seja, existirem assertivas que envolvem ligações entre coleções de bases locais distintas. Neste caso, são geradas VLEs Secundárias que exportam dados das coleções referenciadas por estas ligações. Ainda neste passo, são inferidas as ligações entre as VLEs que exportam dados das coleções envolvidas nas assertivas de junção.

A VM homogênea e as VLEs Primárias possuem elementos primários que representam o mesmo conceito do mundo real, por exemplo, Paciente. As VLEs Secundárias possuem elementos primários que representam o mesmo conceito de algum subelemento complexo da VM, por exemplo, Patologia. Portanto, as VLEs Primárias representam *fragmentos horizontais* (ou *híbridos*, caso não possuam todas as propriedades da VM) e as VLEs Primárias representam *fragmentos verticais* da VM. Assim, o estado da VM (ou a composição da resposta de uma consulta) pode ser obtido apenas por operações de união e junção dos estados das VLEs Primárias e Secundárias, respectivamente (ou dos resultados de subconsultas sobre estas VLEs). Em síntese, as VLEs são geradas para simplificar o processamento de consultas, pois minimizam a heterogeneidade estrutural existente entre os esquemas das bases de dados e a VM.

- **Assertivas de Correspondência entre as VLEs e as bases de dados**

Neste trabalho, não trataremos do problema da tradução local das subconsultas (implementação do algoritmo de tradução local). Entretanto, para fins experimentais, utilizamos nas bases de dados o SGBD Oracle [3], o qual permite definir visões SQL/XML [16] sobre esquemas relacionais e permite executar consultas XQuery sobre estas visões. Assim, as VLEs são definidas localmente como visões SQL/XML e as consultas recebidas pelos serviços web locais são diretamente submetidas aos SGBDs, que atuam como tradutores e retornam os resultados em XML.

Assim, ao final da execução do algoritmo **GeraVMHomogenea,** executamos também o algoritmo **GeraDefinicaoSQL/XML**, proposto em [29], que gera a definição SQL/XML das VLEs sobre as coleções das bases de dados a partir das assertivas da VM heterogênea. É importante destacar que a definição SQL/XML de uma VLE já embute o mapeamento entre o tipo da VLE e o esquema da base de dados. Desta forma, para os nossos experimentos, o conjunto de assertivas das VLEs Primárias e Secundárias com as bases de dados não será necessário. A tradução local de uma consulta sobre uma VLE em uma consulta SQL sobre tabelas relacionais é realizada pelo próprio SGBD, com base na definição SQL/XML. Entretanto, visando atender outros experimentos em que estas assertivas sejam necessárias para a tradução local, nosso algoritmo também gera, a partir das assertivas da VM heterogênea, as assertivas que definem as VLEs em termos dos esquemas das bases de dados (conjunto $\mathcal{Y}_{VLE}$).

- **Grafo da Hierarquia de VLEs**

Ao final da geração das VLEs, é gerado o *Grafo da Hierarquia de VLEs* da VM. Neste grafo direcionado, o nó que representa a VM é chamado de nó-raiz. Os nós diretamente conectados ao nó-raiz representam as VLEs Primárias e os demais nós representam as VLEs Secundárias. As arestas representam as ligações entre as VLEs, exceto para as arestas entre o nó-raiz e os nós que representam as VLEs Primárias. Quando uma aresta representa uma ligação entre VLEs, é incluído à aresta um atributo $[\ell'_i, T'_i]$, com a ligação e o Tipo XML que será utilizado como base na junção das VLEs.

O grafo da hierarquia de VLEs é, portanto, uma representação hierárquica das VLEs da VM homogênea do ponto de vista da reconstrução do estado da VM a partir dos estados destas VLEs. Portanto, veremos a seguir que ele será utilizado para gerar a expressão algébrica do plano de reconstrução da VM (**Φ**). Este grafo também servirá de base para a geração do plano de execução de uma consulta, conforme veremos no Capítulo VI. Na Figura 5.1 é exibido um exemplo de um grafo da hierarquia de VLEs.

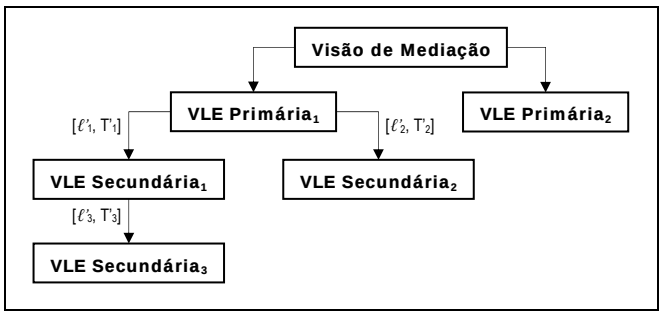


Figura 5.1 – Grafo da Hierarquia de VLEs de uma VM

5.4.2. Geração dos Mapeamentos de Mediação

O processo de geração de uma VM homogênea é finalizado com a geração dos seus mapeamentos de mediação ($\mathcal{M} = <\psi, \Phi>$). Vimos que durante a geração do esquema exportado, são também geradas as assertivas de correspondência de coleção (conjunto ψ) entre a VM homogênea e as VLEs Primárias. Portanto, para complementar estes mapeamentos, é necessário apenas gerar a expressão algébrica (**Φ**) que representa o plano de reconstrução da VM a partir das VLEs do esquema exportado (E).

A expressão algébrica Φ é gerada a partir do grafo de hierarquia de VLEs, em dois passos:

1) ***União das VLE Primárias***, onde a expressão algébrica Φ é inicialmente composta por operações de *Outer Union* ($\ddot{U}_{T,p}$) de todas as VLEs Primárias.

2) ***Junção com as VLEs Secundárias***, onde cada VLE Primária é substituída, na expressão algébrica Φ, por uma (sub)expressão algébrica de junção ($\hat{J}_{T,p}$) da VLE Primária com todas as VLEs Secundárias hierarquicamente ligadas à VLE Primária. Recursivamente, as VLEs Secundárias que possuem ligações com outras VLEs Secundárias são também substituídas por (sub)expressões algébricas de junção ($\hat{J}_{T,p}$).

No primeiro passo, a operação *Outer Union* é utilizada como "padrão" pois consideramos que as VLEs Primárias podem estar em interseção. Desta forma, operações de união simples (*Union*) foram descartadas neste momento[2] pois poderiam resultar em respostas incorretas, com redundâncias. Nas operações algébricas ($\ddot{U}_{T,p}$ e $\hat{J}_{T,p}$), o tipo XML T (utilizado como referência para a construção das respostas) e o predicado p são obtidos conforme se segue: (i) operações de *Outer Union*: T é o tipo da VM e p é inferido da chave da VM; e (ii) operações de junção: dado o atributo $[\ell'_i, T'_i]$ da aresta correspondente à ligação de duas VLEs, T é igual a T'_i e p é inferido de ℓ'_i.

Seja, por exemplo, o grafo da hierarquia de VLEs da Figura 5.1. A expressão algébrica Φ é gerada conforme se segue:

Passo 1: [**Φ**: VM ≡ VLE Primária$_1$ **Ü**$_{...}$ VLE Primária$_2$]

Passo 2: [**Φ**: VM ≡ ((VLE Primária$_1$ **Ĵ**$_{...}$ (VLE Secundária$_1$ **Ĵ**$_{...}$ VLE Secundária$_3$))
Ĵ$_{...}$ VLE Secundária$_2$) **Ü**$_{...}$ VLE Primária$_2$]

Para facilitar a visualização das operações, os atributos T e p foram omitidos.

[2] Na etapa de *otimização global* da metodologia proposta para o processamento de consultas sobre VMs (vide Capítulo VI), veremos que as operações *Union* e *Outer Union* podem ser simultaneamente utilizadas na integração das VLEs Primárias, considerando a existência de interseção ou de disjunção entre estas VLEs.

5.5. Exemplo

Considere novamente o exemplo do "Prontuário Unificado", apresentado no Capítulo III. As informações dos pacientes e dos seus atendimentos serão obtidas de bases de dados heterogêneas, localizadas em hospitais e postos de saúde da cidade. O modelo conceitual desta aplicação de integração de dados é exibido na Figura 5.2.

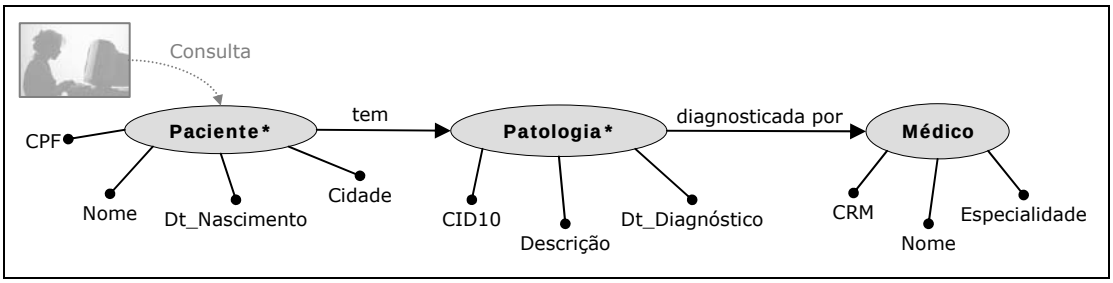


Figura 5.2 – Modelo conceitual da aplicação "Prontuário Unificado"

Nesta seção, veremos como a VM heterogênea **Paciente'$_M$** foi especificada diretamente em termos das bases de dados e como foi transformada na VM homogênea **Paciente$_M$**, que atende aos requisitos desta aplicação.

5.5.1. Especificação da VM Heterogênea Paciente'$_M$

Conforme vimos, a especificação de uma VM heterogênea é composta por três atividades: (i) Definição do esquema de mediação (S); (ii) Definição do esquema local (L); e (iii) Definição dos mapeamentos da VM heterogênea ($\mathcal{Y}$). A seguir, apresentamos detalhes destas atividades, durante a especificação de **Paciente'$_M$**.

- **Definição do Esquema de Paciente'$_M$**

Nesta atividade, o projetista define o nome do elemento primário da VM e o seu tipo XML. No exemplo, o projetista definiu que o elemento primário da VM heterogênea Paciente'$_M$ é Paciente, cujo tipo XML é TPaciente$_M$, exibido na Figura 5.3.

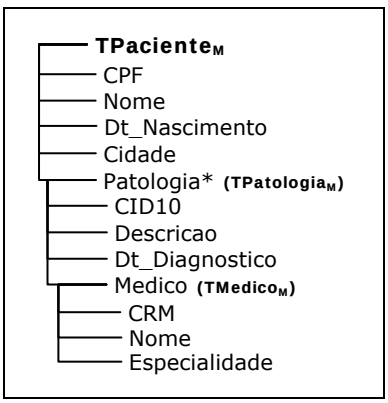


Figura 5.3 – TPaciente$_M$ (Tipo XML de Paciente$_M$)

- **Definição do Esquema Local**

Nesta atividade, o projetista informa quais as fontes que deverão prover dados para compor a VM e, opcionalmente, define as restrições de interseção e as ligações entre coleções de fontes de dados distintas. Para o exemplo, o projetista definiu como fontes de dados as bases *BD1*, *BD2*, *BD3* e *BD4*, cujos esquemas são exibidos na Figura 5.4.

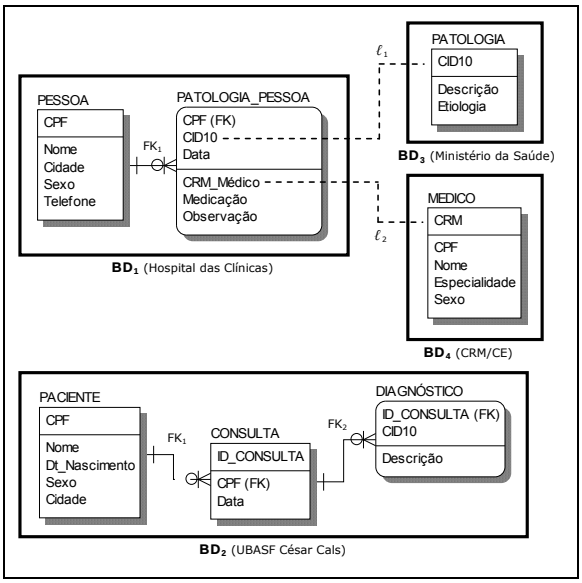


Figura 5.4 – Esquemas das Bases de Dados

A base BD_1 armazena informações de atendimento a pacientes no Hospital das Clínicas. A base BD_2 armazena informações de atendimento a pacientes na UBASF César Cals (posto de saúde). A base BD_3, de propriedade do Ministério da Saúde, armazena informações de patologias. A base BD_4, de propriedade do CRM/CE, armazena informações de médicos credenciados a atender no estado.

Considerando o fato de que um mesmo paciente pode ser atendido tanto no Hospital das Clínicas (BD_1) e como na UBASF César Cals (BD_2), o projetista também definiu uma restrição de interseção entre as coleções que representam as tabelas BD_1:Pessoa e BD_2:Paciente, dada por:

$[r'_1: BD_1\text{:Pessoa} \cap BD_2\text{:Paciente} \neq \varnothing]$

Por fim, ainda foram definidas duas ligações ℓ'_1 e ℓ'_2 (vide Figura 5.4), entre as coleções que representam as tabelas BD_1:Patologia_Pessoa, BD_3:Patologia e BD_4:Medico, dadas por:

$[\ell'_1: \langle BD_1\text{:Patologia\_Pessoa, \{CID10\}, } BD_3\text{:Patologia, \{CID10\}} \rangle]$

$[\ell'_2: \langle BD_1\text{:Patologia\_Pessoa, \{CRM\}, } BD_4\text{:Medico, \{CRM\}} \rangle]$

Observe que as bases BD_3 e BD_4 não possuem coleções-base, ou seja, tabelas com informações de pacientes, mas apenas informações que podem complementar o conteúdo de BD_1:Pessoa.

- **Definição dos Mapeamentos de Paciente'$_M$**

O projetista definiu as assertivas de correspondência de Paciente'$_M$ da seguinte forma:

- Base BD_1:

1) Escolheu a tabela BD_1:Pessoa como coleção-base;

2) Definiu as seguintes ACGs entre Paciente'$_M$ e a coleção-base BD_1:Pessoa:

$[\ \psi'_1: Paciente'_M \supset BD_1:Pessoa\ ]$

$[\ \psi'_2: Paciente'_M[Cidade = \text{'Fortaleza'}] \equiv BD_1:Pessoa\ ]$

3) Realiza o *matching* de $TPaciente_M$ com o tipo da coleção-base BD_1:Pessoa (denotado por $T_{BD_1:Pessoa}$) e com os tipos das coleções ligadas à coleção-base. As ACCs resultantes do processo de *matching* são exibidas na Figura 5.5(a).

- Base BD_2:

1) Escolheu a tabela BD_2:Paciente como coleção-base;

2) Definiu a seguinte ACG entre Paciente'$_M$ e a coleção-base BD_2:Paciente:

$[\ \psi'_3: Paciente'_M \supset BD_2:Paciente\ ]$

3) Realiza o *matching* de $TPaciente_M$ com o tipo da coleção-base BD_2:Paciente (denotado por $T_{BD_2:Paciente}$) e com os tipos das coleções ligadas à coleção-base. As ACCs resultantes do processo de *matching* são exibidas na Figura 5.5(b).

Ao final das três atividades, a VM heterogênea Paciente'$_M$ está especificada, conforme se segue:

Paciente'$_M$ = <**S**, **L**, Υ>, onde:

- **S** = <Paciente, $TPaciente_M$>;

- **L** = <{BD_1, BD_2, BD_3, BD_4}, R'>, onde:

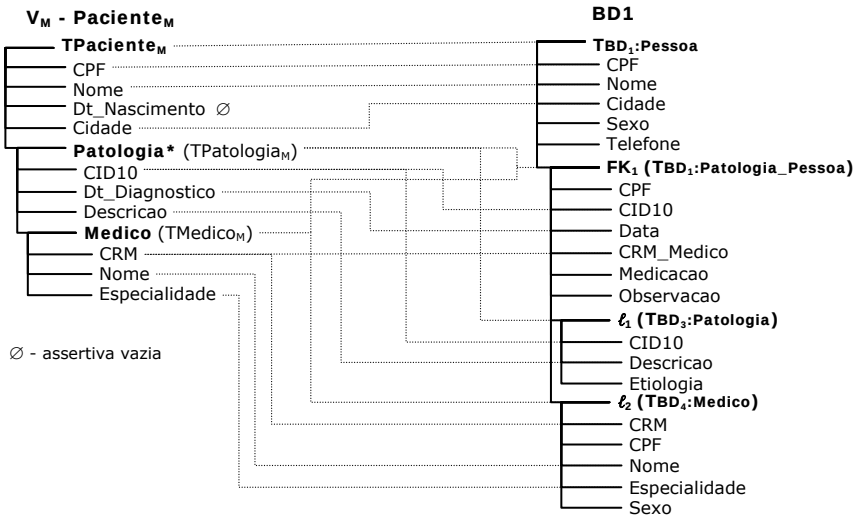
  - R' = { r'_1 , ℓ'_1 , ℓ'_2}

- Υ = <ψ', $\mathcal{A}$ >, onde:

  - ψ' = { ψ'_1 , ψ'_2 , ψ'_3}

  - $\mathcal{A}$ = { $\varphi'_{1.1}$, ..., $\varphi'_{1.14}$, $\varphi'_{3.1}$, ..., $\varphi'_{3.4}$, $\varphi'_{4.1}$, ..., $\varphi'_{4.3}$, $\varphi'_{2.1}$, ..., $\varphi'_{2.9}$ }

(a)



ACGs – ψ'_1: $Paciente_M \supset BD_1$:Pessoa

ψ'_2: $Paciente_M$[Cidade = 'Fortaleza'] $\equiv BD_1$:Pessoa

ACCs de $TPaciente_M$ e TBD_1:Pessoa

$\varphi'_{1.1}$: $TPaciente_M$/CPF ≡ TBD_1:Pessoa/CPF

$\varphi'_{1.2}$: $TPaciente_M$/Nome ≡ TBD_1:Pessoa/Nome

$\varphi'_{1.3}$: $TPaciente_M$/Dt_Nascimento ≡ TBD_1:Pessoa/∅

$\varphi'_{1.4}$: $TPaciente_M$/Cidade ≡ TBD_1:Pessoa/Cidade

$\varphi'_{1.5}$: $TPaciente_M$/Patologia ⊃ TBD_1:Pessoa/FK_1

$\varphi'_{1.6}$: $TPaciente_M$/Patologia ⊃ TBD_1:Pessoa/FK_1/ℓ'_1

ACCs de $TPatologia_M$ e TBD_1:Patologia_Pessoa

$\varphi'_{1.7}$: $TPatologia_M$/CID10 ≡ TBD_1:Patologia_Pessoa/CID10

$\varphi'_{1.8}$: $TPatologia_M$/Dt_Diagnostico ≡ TBD_1:Patologia_Pessoa/Data

$\varphi'_{1.9}$: $TPatologia_M$/Descricao ≡ TBD_1:Patologia_Pessoa/∅

$\varphi'_{1.10}$: $TPatologia_M$/Medico ≡ TBD_1:Patologia_Pessoa/NULL

$\varphi'_{1.11}$: $TPatologia_M$/Medico ≡ TBD_1:Patologia_Pessoa/ℓ'_2

ACCs de $TMedico_M$ e TBD_1:Patologia_Pessoa

$\varphi'_{1.12}$: $TMedico_M$/CRM ≡ TBD_1:Patologia_Pessoa/CRM

$\varphi'_{1.13}$: $TMedico_M$/Nome ≡ TBD_1:Patologia_Pessoa/∅

$\varphi'_{1.14}$: $TMedico_M$/Especialidade ≡ TBD_1:Patologia_Pessoa/∅

ACCs de $TPatologia_M$ e TBD_3:Patologia

$\varphi'_{3.1}$: $TPatologia_M$/CID10 ≡ TBD_3:Patologia/CID10

$\varphi'_{3.2}$: $TPatologia_M$/Dt_Diagnostico ≡ TBD_3:Patologia/∅

$\varphi'_{3.3}$: $TPatologia_M$/Descricao ≡ TBD_3:Patologia/Descricao

$\varphi'_{3.4}$: $TPatologia_M$/Medico ≡ TBD_3:Patologia/∅

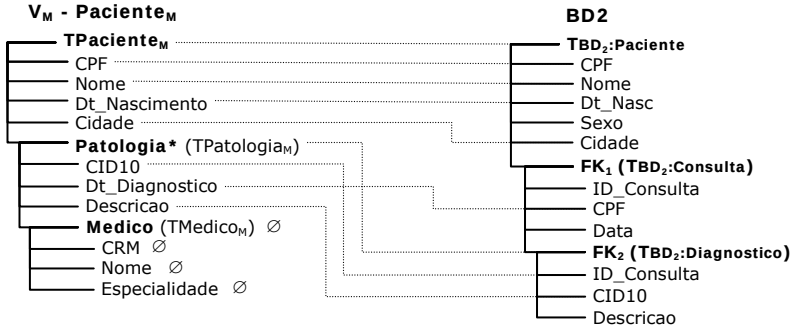
ACCs de $TMedico_M$ e TBD_4:Medico

$\varphi'_{4.1}$: $TMedico_M$/CRM ≡ TBD_4:Medico/CRM

$\varphi'_{4.2}$: $TMedico_M$/Nome ≡ TBD_4:Medico/Nome

$\varphi'_{4.3}$: $TMedico_M$/Especialidade ≡ TBD_4:Medico/Especialidade

(b)



ACG – ψ'_2: $Paciente_M \supset BD_2$:Paciente

ACCs de $TPaciente_M$ e TBD_2:Paciente

$\varphi'_{2.1}$: $TPaciente_M$/CPF ≡ TBD_2:Paciente/CPF

$\varphi'_{2.2}$: $TPaciente_M$/Nome ≡ TBD_2:Paciente/Nome

$\varphi'_{2.3}$: $TPaciente_M$/Dt_Nascimento ≡ TBD_2:Paciente/Dt_Nascimento

$\varphi'_{2.4}$: $TPaciente_M$/Cidade ≡ TBD_2:Paciente/Cidade

$\varphi'_{2.5}$: $TPaciente_M$/Patologia ⊃ TBD_2:Paciente/FK_1/FK_2

ACCs de $TPatologia_M$ e TBD_2:Diagnostico

$\varphi'_{2.6}$: $TPatologia_M$/CID10 ≡ TBD_2:Diagnostico/CID10

$\varphi'_{2.7}$: $TPatologia_M$/Dt_Diagnostico ≡ TBD_2:Diagnostico/FK_2^{-1}/Data

$\varphi'_{2.8}$: $TPatologia_M$/Descricao ≡ TBD_2:Diagnostico/Descricao

$\varphi'_{2.9}$: $TPatologia_M$/Medico ≡ TBD_2:Diagnostico/∅

Figura 5.5 – (a) Assertivas entre $TPaciente_M$ e BD_1, BD_3 e BD_4 ; (b) Assertivas entre $TPaciente_M$ e BD_2

5.5.2. Geração da VM Homogênea Paciente$_M$

Conforme vimos, a geração de uma VM homogênea consiste de três atividades: (i) Geração do esquema de mediação (S); (ii) Geração do esquema exportado (E); e (iii) Geração dos mapeamentos da VM homogênea ($\mathcal{M}$). A seguir, apresentamos detalhes destas atividades, durante o processo de geração de **Paciente$_M$**.

- **Geração do esquema de mediação de Paciente$_M$**

Conforme vimos, o esquema de mediação de uma VM homogênea é o mesmo da VM heterogênea a partir da qual a VM homogênea será gerada. Assim, este passo consiste apenas da atribuição do esquema de Paciente'$_M$ como esquema de Paciente$_M$. Para o exemplo dado, o projetista definiu o seguinte esquema de mediação: <Paciente, TPaciente$_M$>, onde Paciente é o elemento primário e TPaciente$_M$ é o tipo de Paciente, como foi mostrado na Figura 5.3.

- **Geração do Esquema Exportado de Paciente$_M$**

  - **Geração das VLEs Primárias**

As VLEs Primárias são geradas analisando-se as assertivas de Paciente'$_M$ com cada base de dados. Apenas as bases onde foram identificadas coleções-base serão consideradas pelo algoritmo para a geração das VLEs Primárias, ou seja, bases de dados que possuem coleções para as quais foram definidas Assertivas de Correspondência Global ou de Coleção (ACGs) com Paciente'$_M$.

Considere o conjunto de assertivas $\mathcal{A}$ de Paciente'$_M$ exibido na Figura 5.5. Para a base de dados BD$_1$, foi identificada a coleção-base *BD$_1$:Pessoa* e definida a ACG *[ψ'_1: Paciente'$_M$ $\supset$ BD$_1$:Pessoa]*. Assim, é gerada a partir do conjunto de assertivas $\mathcal{A}$ a VLE Primária **Paciente$_1$**, cujo esquema é dado por S$_1$=<Paciente, TPaciente$_1$>. Além disso, é gerada uma ACG entre Paciente$_M$ e Paciente$_1$ dada por *[ψ_1: Paciente$_M$ $\supset$ Paciente$_1$]*, conforme exibido na Figura 5.6(a).

Note que:

- Quando comparado ao tipo TPaciente$_M$, o tipo TPaciente$_1$ não contém os elementos Paciente/Dt_Nascimento, Paciente/Patologia/Descricao, Paciente/Patologia/Medico/Nome e Paciente/Patologia/Medico/Especialidade, pois não existem propriedades em BD$_1$ correspondentes a estes elementos (correspondências vazias ($\varnothing$)).

- Todos os demais elementos de $TPaciente_1$ possuem a mesma estrutura dos elementos correspondentes em $TPaciente_M$ e, como os elementos primários de $Paciente_1$ e $Paciente_M$ representam o mesmo conceito do mundo real, a VLE $Paciente_1$ é considerada um fragmento horizontal de $Paciente_M$.

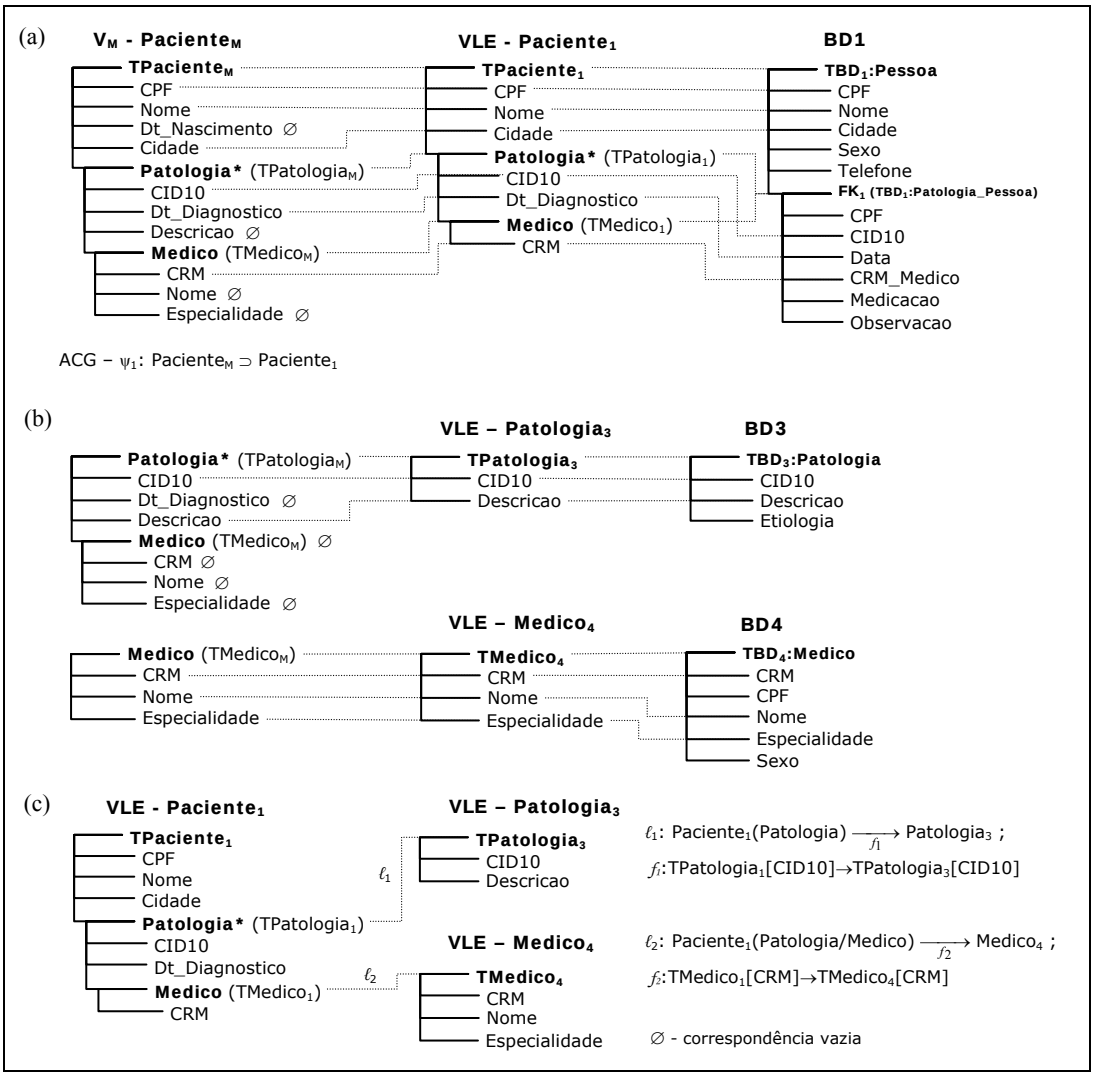


Figura 5.6 – (a) VLE $Paciente_1$ e ACG com $Paciente_M$; (b) VLEs $Patologia_3$ e $Medico_4$; (c) Ligações entre $Paciente_1$, $Patologia_3$ e $Medico_4$

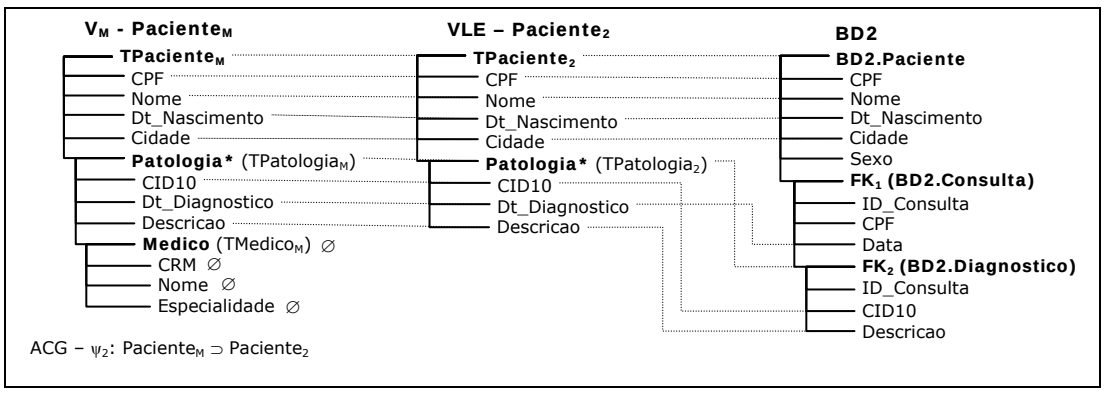


Figura 5.7 – VLE $Paciente_2$ e ACG com $Paciente_M$

O processo de geração é repetido para a base BD_2, onde também foi identificada a coleção-base *BD2:Paciente* e definida a ACG *[ψ'2: Paciente'M ⊃ BD2:Paciente]*. A VLE Primária **Paciente2**, cujo esquema é dado por S_2=<Paciente, $TPaciente_2$>, é gerada a partir do conjunto de assertivas $\mathcal{A}$. Assim como $Paciente_1$, $Paciente_2$ também é considerada um fragmento horizontal de $Paciente_M$. Na geração de $Paciente_2$ é gerada uma ACG entre $Paciente_M$ e $Paciente_2$ dada por $[\psi_2: Paciente_M \supset Paciente_2]$, conforme mostra a Figura 5.7.

Ainda neste passo, a partir das restrições de interseção entre as coleções-base são inferidas as restrições de interseção entre as VLEs Primárias. O projetista, ao especificar a VM heterogênea $Paciente'_M$, definiu a seguinte restrição de interseção:

$[r'_1: BD_1:Pessoa \cap BD_2:Paciente \neq \varnothing]$

A partir de r'_1, o algoritmo infere a seguinte restrição de interseção entre $Paciente_1$ e $Paciente_2$:

$[r_1: Paciente_1 \cap Paciente_2 \neq \varnothing]$.

- **Geração das VLEs Secundárias**

Após a geração da VLE Primária de cada base, o algoritmo analisa se, dentre as assertivas definidas para aquela base, existem assertivas de junção. Para o exemplo, existem duas assertivas de junção (vide Figura 5.5(a)):

$[\varphi'_{1.6}: TPaciente_M/Patologia \supset TBD_1:Pessoa/FK_1/\ell'_1]$

$[\varphi'_{1.11}: TPatologia_M/Medico \equiv TBD_1:Patologia\_Pessoa/\ell'_2]$

Como as ligações ℓ'_1 e ℓ'_2 referenciam, respectivamente, coleções das bases BD_3 e BD_4, são geradas as VLEs Secundárias $Patologia_3$ e $Medico_4$ sobre as bases BD_3 BD_4, respectivamente.

Note que:

- Conforme o processo de definição de assertivas (*matching*), que obriga a definição de assertivas para tipos complexos envolvidos em assertivas de junção (*especifica completamente*), foram definidas assertivas entre: (i) o tipo $TPatologia_M$ e a coleção BD_3:Patologia ($\varphi'_{3.1}$ a $\varphi'_{3.4}$); e (ii) o tipo entre $TMedico_M$ e a coleção BD_4:Medico ($\varphi'_{4.1}$ a $\varphi'_{4.3}$). A partir destas assertivas, são geradas as VLEs Secundárias $Patologia_3$ e $Medico_4$, conforme mostrado na Figura 5.6(b).

- A partir da ligação ℓ'_1 entre as coleções BD_1:Patologia_Pessoa e BD_3:Patologia é inferida uma ligação ℓ_1 entre a VLE Primária $Paciente_1$ e a VLE Secundária $Patologia_3$; e

a partir da ligação ℓ'_2 entre as coleções BD_1:Patologia_Pessoa e BD_4:Medico é inferida uma ligação ℓ_2 entre a VLE Primária $Paciente_1$ e a VLE Secundária $Medico_4$, conforme mostrado na Figura 5.6(c).

- **Geração do grafo da hierarquia de VLEs de $Paciente_M$**

Por fim, considerando as VLEs Primárias e Secundárias geradas e as ligações entre as VLEs, é gerado o grafo da hierarquia de VLEs de $Paciente_M$, conforme exibido na Figura 5.8.

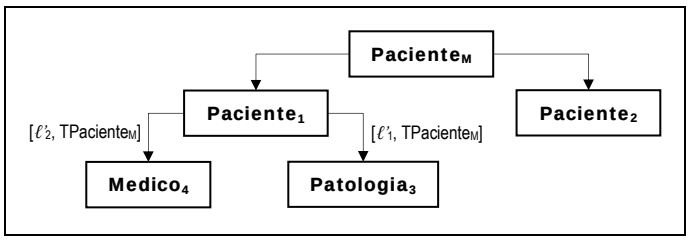


Figura 5.8 – Grafo da Hierarquia de VLEs de $Paciente_M$

O nó [$Paciente_M$] é o nó-raiz, o qual representa a VM. A partir das ACGs ψ_1 e ψ_2 de $Paciente_M$ com as VLEs Primárias, são incluídos os nós [$Paciente_1$] e [$Paciente_2$] ligados ao nó-raiz. A partir das ligações ℓ_1 e ℓ_2, são incluídos os nós [$Patologia_3$] e [$Medico_4$] ligados ao nó [$Paciente_1$]. Ainda com base nestas ligações, são gerados os atributos [ℓ_1, $TPaciente_M$] e [ℓ_2, $TPaciente_M$] das arestas entre os nós [$Paciente_1$], [$Patologia_3$] e [$Medico_4$].

As definições SQL/XML das VLEs (vide Figuras 5.9, 5.10 e 5.11) são geradas pelo algoritmo **GeraDefinicaoSQL/XML**, a partir das assertivas ($\mathcal{A}$) da VM heterogênea $Paciente'_M$.

```
CREATE VIEW Paciente1 … AS
SELECT XMLElement("Paciente",
          XMLForest(P.CPF as "CPF"),
          XMLForest(P.Nome as "Nome"),
          XMLForest(P.Cidade as "Cidade"),
         (SELECT XMLAgg( XMLElement("Patologia",
                         XMLForest(PP.CID10 as "CID10"),
                         XMLForest(PP.Data as "Dt_Diagnostico"),
                         XMLElement("Medico",
                          XMLForest(PP.CRM_Medico as "CRM") ) ) )
            FROM BD1.Patologia_Pessoa PP
           WHERE PP.CPF = P.CPF) )
  FROM BD1.Pessoa P
```

Figura 5.9 – Definição SQL/XML da VLE $Paciente_1$, sobre a base BD1

```
CREATE VIEW Patologia3 … AS
SELECT XMLElement("Patologia",
        XMLForest(P.CID10 as
                  "CID10"),
        XMLForest(P.Descricao as
                  "Descricao"))
FROM BD3.Patologia P
```

```
CREATE VIEW Medico4 … AS
SELECT XMLElement("Medico",
         XMLForest(M.CRM as "CRM"),
         XMLForest(M.Nome as "Nome"),
         XMLForest(M.Especialidade as
                   "Especialidade"))
FROM BD4.Medico M
```

Figura 5.10 – Definição SQL/XML das VLEs $Patologia_3$ e $Medico_4$, sobre as bases BD3 e BD4, respectivam.

```
CREATE VIEW Paciente2 … AS
SELECT XMLElement("Paciente",
          XMLForest(P.CPF as "CPF"),
          XMLForest(P.Nome as "Nome"),
          XMLForest(P.Dt_Nascimento as "Dt_Nascimento"),
          XMLForest(P.Cidade as "Cidade"),
         (SELECT XMLAgg( XMLElement("Patologia",
                         XMLForest(D.CID10 as "CID10"),
                         XMLForest(C.Data as "Dt_Diagnostico"),
                         XMLForest(D.Descricao as "Descricao") ) )
            FROM BD2.Consulta C, BD2.Diagnostico D
           WHERE C.CPF = P.CPF
             AND C.ID_Consulta = D.ID_Consulta) )
FROM BD2.Paciente P
```

Figura 5.11 – Definição SQL/XML da VLE $Paciente_2$, sobre a base BD2

Na Figura 5.12 é apresentada a arquitetura de três níveis de esquemas da VM homogênea $Paciente_M$, após a geração das VLEs. Resumindo, a base primária BD_1 exporta a VLE Primária $Paciente_1$, do tipo $TPaciente_1$, a qual possui ligação com duas VLEs Secundárias: $Patologia_3$ e $Medico_4$. A VLE Secundária $Patologia_3$ pode ser utilizada para obter informações complementares de $Paciente_1$/Patologia, como o elemento $Paciente_M$/Patologia/Descricao. Já a VLE Secundária $Medico_4$ pode ser utilizada para obter informações complementares de $Paciente_1$/Patologia/Medico, como os elementos $Paciente_M$/Patologia/Medico/Nome e $Paciente_M$/Patologia/Medico/Especialidade. A base primária BD_2 exporta a VLE $Paciente_2$, do tipo $TPaciente_2$, a qual não possui ligação com VLEs Secundárias.

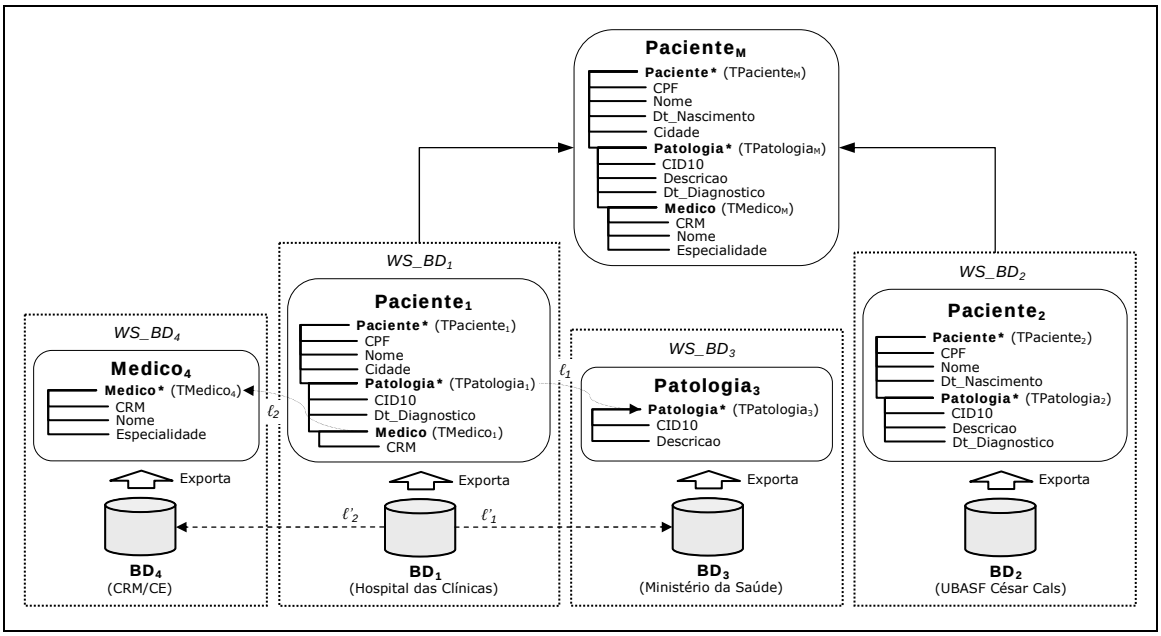


Figura 5.12 – VM $Paciente_M$; VLEs Primárias $Paciente_1$ e $Paciente_2$; e VLEs Secundárias $Patologia_3$ e $Medico_4$

- **Geração dos Mapeamentos de Paciente_M**

Os mapeamentos de mediação de Paciente_M são dados por $\mathcal{M} = <\psi, \Phi>$. Como as ACGs (conjunto $\psi=\{\psi_1, \psi_2\}$) já foram criadas durante a geração das VLEs, é necessário apenas gerar a expressão algébrica Φ para concluir a geração de Paciente_M.

A expressão algébrica **Φ** é gerada em dois passos, a partir do grafo da hierarquia de VLEs exibido na Figura 5.8.

Passo 1:

$$[\Phi: \textbf{Paciente}_M \equiv \textbf{Paciente}_1 \ddot{\cup}_{TPaciente_M,Paciente/CPF=Paciente/CPF} \textbf{Paciente}_2]$$

Passo 2:

$$[\Phi: \textbf{Paciente}_M \equiv ((\textbf{Paciente}_1 \hat{J}_{TPaciente_M,Paciente/Patologia/CID10=Patologia/CID10} \textbf{Patologia}_3) \hat{J}_{TPaciente_M,Paciente/Patologia/Medico/CRM=Medico/CRM} \textbf{Medico}_4) \ddot{\cup}_{TPaciente_M,Paciente/CPF=Paciente/CPF} \textbf{Paciente}_2]$$

Uma descrição intuitiva da semântica desta expressão $\textbf{Paciente}_M$ é a seguinte:

"Para obter todos os pacientes de Paciente_M, ou seja, o estado de Paciente_M, realize as seguinte operações:

1. Consulte todos os pacientes da base BD_1, através da VLE Paciente_1;
2. Na base BD_3, através da VLE Patologia_3, obtenha mais informações das patologias que foram retornadas no resultado da operação 1;
3. Na base BD_4, através da VLE Medico_4, obtenha mais informações dos médicos que foram retornados no resultado da operação 1;
4. Complemente (por junção) as informações de patologias e médicos presentes no resultado do passo 1 com os resultados dos passos 2 e 3;
5. Consulte todos os pacientes da base BD_2, através da VLE Paciente_2;
6. Realize a integração (por *Outer Union*) das informações dos pacientes obtidos no resultado das operações 4 e 5."

5.6. Algoritmos

Nesta seção, é apresentado o algoritmo **GeraVMHomogenea**, que transforma a especificação de uma VM heterogênea em uma VM homogênea equivalente. O algoritmo também gera as VLEs, as restrições de interseção e as ligações entre as VLEs.

Algoritmo: Gera_VM_Homogenea

Entrada: V_M' (V_M' é a especificação de uma VM heterogênea, dada por $V_M' = <S, L, \mathcal{Y}>$, onde:

$S = <e, T>$, $L = <\{L_1, ..., L_n\}, R'>$ e $\mathcal{Y} = <\psi', \mathcal{A}>$).

Saída: V_M (V_M é uma visão de mediação homogênea, dada por $V_M = <S, E, \mathcal{M}>$, onde:

$S = <e, T>$, $E = <\{V_{LE_1}, ..., V_{LE_n}\}, R>$ e $\mathcal{M} = <\psi, \Phi>$).

início

$VLEs_{Prim} \leftarrow \emptyset$; $VLEs_{Sec} \leftarrow \emptyset$; $\psi \leftarrow \emptyset$; $R \leftarrow \emptyset$; $Ligs \leftarrow \emptyset$;

// Gerando as VLEs Primárias com base no conjunto ψ' (Assertivas de Correspondência Global ou de Coleção (ACGs)).

para cada $y' \in \psi'$ **faça** // y' é da forma $[V_M \equiv C_i]$, $[V_M \supset C_i]$ ou $[V_M[Exp] \equiv C_i]$, onde C_i é uma coleção-base
// cujo esquema é dado por $S_i = <e_i, TC_i>$

$C_i \leftarrow$ **Obtem_Colecao_Base(**y'**);**

// Obtém do conjunto $\mathcal{A}$ (Assertivas de Correspondência de Caminho (ACCs) as assertivas que especificam
// completamente o tipo T da VM em função de TC_i.

$\mathcal{A}_{[T\&TC_i]} \leftarrow$ **Obtem_ACs_Tipos(**$\mathcal{A}$, T, TC_i**);**

// Gerando a VLE Primária

$<T_{V_{[Ci]}}, \mathcal{A}_{[T_{V_{[Ci]}}\&TC_i]}> \leftarrow$ **Gera_Tipo_E_Assertivas_VLE(**$\mathcal{A}_{[T\&TC_i]}$**);**

$V_{[Ci]} \leftarrow$ **Cria_VLE(**e, $T_{V_{[Ci]}}$, $\mathcal{A}_{[T_{V_{[Ci]}}\&TC_i]}$**);**

$VLEs_{Prim} \leftarrow VLEs_{Prim} \cup \{V_{[Ci]}\}$;

// Gerando, a partir de y', a ACG entre a VM e a VLE Primária $V_{[Ci]}$. Para isto, a coleção C_i é substituída por $V_{[Ci]}$.

$y \leftarrow$ **Gera_ACG_VM_VLE(**y', C_i, $V_{[Ci]}$**);** // Ex: ψ': $[V_M \equiv C_i] \rightarrow \psi$: $[V_M \equiv V_{[Ci]}]$

$\psi \leftarrow \psi \cup \{y\}$;

// Gerando as VLEs Secundárias ligadas à VLE Primária $V_{[Ci]}$.

$<VLEs_{Sec}, Ligs_{[VLEsSec]}> \leftarrow$ **Gera_VLEs_Secundarias(**C_i, $V_{[Ci]}$, $\mathcal{A}_{[T\&TC_i]}$**);**

$VLEs_{Sec} \leftarrow VLEs_{Sec} \cup VLEs_{Sec}$;

$Ligs \leftarrow Ligs \cup Ligs_{[VLEsSec]}$;

// Gerando o esquema consolidado da VLE Primária $V_{[Ci]}$ (definição no Capítulo VI).

Gera_Esquema_Consolidado($V_{[Ci]}$, $VLEs_{Sec}$, $Ligs_{[VLEsSec]}$**);**

fim para;

// Gerando as restrições de interseção entre as VLEs Primárias e o esquema exportado E.

$R \leftarrow$ **Gera_Restricoes_Intersecao(**R', $VLEs_{Prim}$**);**

$E \leftarrow <VLEs_{Prim} \cup VLEs_{Sec}, R \cup Ligs>$;

// Gerando grafo da hierarquia de VLEs

$G_{Hieraq} \leftarrow$ **Gera_Grafo_Hierarquia_VLEs(**ψ, Ligs**);**

// Gerando a expressão algébrica de reconstrução de V_M

$\Phi \leftarrow$ **Gera_Expressao_Algebrica_Reconstrucao(**G_{Hieraq}**);**

// Gerando os mapeamentos de mediação $\mathcal{M}$

$\mathcal{M} \leftarrow <\psi, \Phi>$;

retorna $V_M = <S, E, \mathcal{M}>$;

fim.

Algoritmo: Gera_VLEs_Secundarias

Entrada: C_i, $V_{[Ci]}$, $\mathcal{A}_{[T\&TC_i]}$ (C_i é uma coleção cujo esquema é dado por C_i=<e_i, TC_i> ; $V_{[Ci]}$ é uma VLE definida em termos da coleção C_i ; $\mathcal{A}_{[T\&TC_i]}$ é o conjunto de assertivas que especificam completamente o tipo T em termos de TC_i).

Saída: $VLEs_{Sec}$, $Ligs_{[VLEsSec]}$ ($VLEs_{Sec}$ é o conjunto de VLEs Secundárias geradas e $Ligs_{[VLEsSec]}$ é o conjunto de ligações entre as VLEs Secundárias geradas).

início

$VLEs_{Sec} \leftarrow \emptyset$; $Ligs_{[VLEsSec]} \leftarrow \emptyset$;

// Obtendo o conjunto de Assertivas de Junção do conjunto $\mathcal{A}_{[T\&TC_i]}$

ACCsJuncao ← **Obtem_Assertivas_Juncao(**$\mathcal{A}_{[T\&TC_i]}$**)**;

para cada $\varphi' \in$ ACCsJuncao **faça** // Onde: φ':[T'/e'] ≡ [T_i/δ/ℓ'] ou φ':[T'/e'] ⊃ [T_i/δ/ℓ'] ;

// ℓ':$C_i(\lambda) \xrightarrow{f} C_S$; C_S=<e_S,TC_S> e f:$T\lambda\{a_1, ..., a_n\} \rightarrow TC_S\{b_1, ..., b_n\}$.

// Obtém de $\mathcal{A}_{[T\&TC_i]}$ o conjunto de assertivas que especificam completamente o tipo $T_{e'}$ em função de TC_S.

$\mathcal{A}_{[T_{e'}\&TC_S]}$ ← **Obtem_ACs_Tipos(**$\mathcal{A}_{[T\&TC_i]}$, $T_{e'}$, TC_S**)**;

// Gerando a VLE Secundária

<$T_{V_{[CS]}}$, $\mathcal{A}_{[T_{V_{[CS]}}\&TC_S]}$> ← **Gera_Tipo_E_Assertivas_VLE(**$\mathcal{A}_{[T_{e'}\&TC_S]}$**)**;

$V_{[CS]}$ ← **Cria_VLE(**e', $T_{V_{[CS]}}$, $\mathcal{A}_{[T_{V_{[CS]}}\&TC_S]}$**)**;

// Gerando uma ligação entre as VLEs $V_{[Ci]}$ e $V_{[CS]}$.

ℓ ← **Gera_Ligacao_VLEs(**φ', $V_{[Ci]}$, $V_{[CS]}$**)**;

$Ligs_{[VLEsSec]} \leftarrow Ligs_{[VLEsSec]} \cup \{\ell\}$;

$VLEs_{Sec} \leftarrow VLEs_{Sec} \cup \{V_{[CS]}\}$;

// Gerando as VLEs Secundárias ligadas à VLE Secundária $V_{[CS]}$.

<$VLEs_{Sec}$', $Ligs_{[VLEsSec]}$'> ← **Gera_VLEs_Secundarias(**C_S, $V_{[CS]}$, $\mathcal{A}_{[T_{e'}\&TC_S]}$**)**;

$Ligs_{[VLEsSec]} \leftarrow Ligs_{[VLEsSec]} \cup Ligs_{[VLEsSec]}'$;

$VLEs_{Sec} \leftarrow VLEs_{Sec} \cup VLEs_{Sec}'$;

fim para;

retorna $VLEs_{Sec}$, $Ligs_{[VLEsSec]}$;

fim.

Algoritmo: Gera_Tipo_E_Assertivas_VLE

Entrada: $\mathcal{A}_{[T_1\&T_2]}$ ($\mathcal{A}_{[T_1\&T_2]}$ é o conjunto de assertivas que especificam completamente o tipo T_1 em termos de T_2).

Saída: T_{VLE}, $\mathcal{A}_{[T_{VLE}\&T_2]}$ (T_{VLE} é o tipo da VLE, onde T_{VLE} é um tipo restrito de T_1 ; $\mathcal{A}_{[T_{VLE}\&T_2]}$ é o conjunto de assertivas que especificam completamente o tipo T_{VLE} em termos de T_2).

início

T_{VLE} ← **Cria_Tipo_Vazio()**;

T_{VLE}.**Tipo_Pai_Fragmento(**T_1**)**; // Determina que T_{VLE} é um fragmento de T_1.

$\mathcal{A}_{[T_{VLE}\&T_2]} \leftarrow \emptyset$;

para cada $\varphi' \in \mathcal{A}_{[T_1\&T_2]}$ **faça**

se φ':[T_1/e] ≡ [T_2/δ] **e ehTipo_Simples(**T_e**) e** $\delta \neq \emptyset$ **então**

// Adiciona ao tipo T_{VLE} a propriedade e, cujo tipo é T_e.

Adiciona_Propriedade(T_{VLE}, e, T_e**)**;

φ'' ← [T_{VLE}/e] ≡ [T_2/δ];

$\mathcal{A}_{[T_{VLE}\&T_2]} \leftarrow \mathcal{A}_{[T_{VLE}\&T_2]} \cup \{\varphi''\}$;

fim se;

se (φ':[T_1/e] ≡ [T_2/δ] **ou** φ':[T_1/e] ⊃ [T_2/δ]) **e ehTipo_Complexo(**T_e**) e** $\delta \neq \varnothing$ **então**

// Obtém de $\mathcal{A}_{[T_1 \& T_2]}$ o conjunto de assertivas que especificam completamente o tipo T_e em função do tipo de δ (T_δ).

$\mathcal{A}_{[T_e \& T_\delta]}$ ← **Obtem_ACs_Tipos(**$\mathcal{A}_{[T_1 \& T_2]}$, T_e, T_δ**);**

<$T_{e'}$, $\mathcal{A}_{[T_{e'} \& T_\delta]}$> ← **Gera_Tipo_E_Assertivas_VLE(**$\mathcal{A}_{[T_e \& T_\delta]}$**);**

Adiciona_Propriedade(T_{VLE}, e, $T_{e'}$**);**

se φ':[T_1/e] ≡ [T_2/δ] **então**

φ'' ← [T_{VLE}/e] ≡ [T_2/δ];

senão

φ'' ← [T_{VLE}/e] ⊃ [T_2/δ];

fim se;

$\mathcal{A}_{[T_{VLE} \& T_2]}$ ← $\mathcal{A}_{[T_{VLE} \& T_2]}$ ∪ $\mathcal{A}_{[T_{e'} \& T_\delta]}$ ∪ {φ''};

fim se;

se (φ':[T_1/e] ≡ [T_2/δ/ℓ'] **ou** φ':[T_1/e] ⊃ [T_2/δ/ℓ']) **e ehTipo_Complexo(**T_e**) então**

$\mathcal{A}_{[T_e \& T_\delta]}$ ← **Obtem_ACs_Tipos(**$\mathcal{A}_{[T_1 \& T_2]}$, T_e, T_δ**);**

<$T_{e'}$, $\mathcal{A}_{[T_{e'} \& T_\delta]}$> ← **Gera_Tipo_E_Assertivas_VLE(**$\mathcal{A}_{[T_e \& T_\delta]}$**);**

Adiciona_Propriedade(T_{VLE}, e, $T_{e'}$**);**

se φ':[T_1/e] ≡ [T_2/δ/ℓ'] **então**

φ'' ← [T_{VLE}/e] ≡ [T_2/δ/ℓ'];

senão

φ'' ← [T_{VLE}/e] ⊃ [T_2/δ/ℓ'];

fim se;

$\mathcal{A}_{[T_{VLE} \& T_2]}$ ← $\mathcal{A}_{[T_{VLE} \& T_2]}$ ∪ $\mathcal{A}_{[T_{e'} \& T_\delta]}$ ∪ {φ''};

fim se;

fim para;

retorna T_{VLE} , $\mathcal{A}_{[T_{VLE} \& T_2]}$;

fim.

Algoritmo: Gera_Ligacao_VLEs

Entrada: φ', V_1, V_2 (φ' é uma assertiva de junção com uma ligação ℓ' entre as coleções C_1 e C_2 ; V_1 e V_2 são VLEs definidas em termos das coleções C_1 e C_2).

Saída: ℓ (ℓ é uma ligação entre as VLEs V_1 e V_2).

início

// Consideremos: φ':[T'/e'] ≡ [T_i/δ/ℓ'] ou φ':[T'/e'] ⊃ [T_i/δ/ℓ'] ; ℓ':$C_i(\lambda) \xrightarrow[f]{} C_S$; C_i=<e_i,TC_i>; C_S=<e_S,TC_S>

// e f':Tλ{a'$_1$, ..., a'$_n$} → TC_S{b'$_1$, ..., b'$_n$}.

// Gerando a ligação ℓ a partir de ℓ'.

T_{V_1} ← **Obtem_Tipo(**V_1**);** T_{V_2} ← **Obtem_Tipo(**V_2**);**

f ← **[**T_{V_1}**{**a_1, ..., a_n**}** → T_{V_2}**{**b_1, ..., b_n**}];** // Onde a_1, ...a_n e b_1, ...b_n são chaves de T_{V_1} e T_{V_2}.

λ ← **Obtem_Caminho_Tipo(**TC_i , $T_{e'}$**);**

ℓ ← [$V_1(\lambda) \xrightarrow[f]{} V_2$];

retorna ℓ;

fim.

Algoritmo: Gera_Grafo_Hierarquia_VLEs

Entrada: ACGs, Ligs (ACGs é o conjunto de assertivas de correspondência global ou de coleção entre a VM e as VLEs geradas e Ligs é o conjunto de ligações entre as VLEs geradas).

Saída: $G_{Hierarq}$ ($G_{Hierarq}$ é o grafo da hierarquia de VLEs).

início

// Gerando o no_raiz de G_{Hieraq}

no_raiz ← **Gera_No_Raiz_Grafo_Hieraquia(**V_M**);**

// Adicionando os nós que representam as VLEs Primárias.

para cada $\psi \in$ ACGs **faça** // ψ é da forma [$V_M \equiv V_{[Ci]}$], [$V_M \supset V_{[Ci]}$] ou [V_M[Exp] $\equiv V_{[Ci]}$].

$V_{[Ci]}$ ← **Obtem_VLE(**ψ**);**

no_vle_primaria ← **Gera_No_VLE(**$V_{[Ci]}$**);**

no_raiz.**Adiciona_No_Filho(**no_vle_primaria**);**

fim para;

// Adicionando os nós que representam as VLEs Secundárias.

para cada $\ell \in$ Ligs **faça** // ℓ é da forma ℓ: [$V_1(\lambda) \xrightarrow[f]{} V_2$], onde f:Tλ{a_1, ..., a_n} → TC_S{b_1, ..., b_n}.

V_1 ← **Obtem_VLE_Origem(**ℓ**);**

// Realiza, a partir de no_raiz, busca em largura pelo nó que representa a VLE V_1.

no_vle_origem ← **Busca_No_VLE(**no_raiz, V_1**);**

V_2 ← **Obtem_VLE_Destino(**ℓ**);**

no_vle_destino ← **Gera_No_VLE(**V_2**);**

no_vle_origem.**Adiciona_No_Filho(**no_vle_destino**);**

// Adicionando o atributo [ℓ, T] à aresta entre os nós no_vle_origem e no_vle_destino.

T_{V_1} ← **Obtem_Tipo(**V_1**);**

Adiciona_Atributo_Aresta(no_vle_origem, no_vle_destino,

[ℓ, T_{V_1}.**Obtem_Tipo_Pai_Fragmento()**]**);**

fim para;

$G_{Hierarq}$ ← **Gera_Grafo(**no_raiz**);**

retorna $G_{Hierarq}$;

fim.

Capítulo VI

Processamento de Consultas sobre Visões de Mediação

Neste capítulo propomos uma metodologia para o processamento de consultas sobre visões de mediação homogêneas. A metodologia está dividida em três etapas: localização dos dados (identificação das VLEs relevantes); decomposição da consulta (em subconsultas sobre as VLEs relevantes); e otimização global, onde são adotadas heurísticas que objetivam aumentar a eficiência do plano de execução da consulta.

6.1. Introdução

De acordo com [32], processamento de consultas é o processo pelo qual uma consulta declarativa é traduzida em operações de mais baixo nível para a manipulação de dados. Os autores apresentam uma metodologia para o processamento de consultas sobre bases relacionais distribuídas que consiste de quatro etapas:

(i) *Decomposição da consulta*, onde a consulta é traduzida em uma consulta algébrica sobre relações globais. Nesta etapa é realizada também uma validação sintática e semântica da consulta e a sua simplificação (eliminação de predicados redundantes);

(ii) *Localização dos dados*, onde as referências às relações globais são substituídas na consulta algébrica pelos respectivos fragmentos que compõem aquelas relações. A substituição é realizada utilizando informações de fragmentação e distribuição dos dados. É gerada, então, uma consulta algébrica sobre os fragmentos;

(iii) *Otimização global*, onde o objetivo é encontrar, dentre as diversas estratégias possíveis para execução da consulta, aquela estratégia que seja mais próxima da estratégia ótima. Para isto, é utilizada um função de custo. Em ambientes distribuídos, o fator considerado mais significativo para esta função é, normalmente, o custo de comunicação;

(iv) *Otimização local*, onde subconsultas reescritas sobre os fragmentos são otimizadas pelos sistemas locais. Estes sistemas utilizam informações locais sobre a organização física dos dados (como a disponibilidade de índices, etc) para determinar qual a estratégia de execução da subconsulta mais próxima da estratégia ótima.

Em [19] é proposta uma metodologia, adaptada de [32], para o processamento de consultas XQuery sobre bases de dados XML distribuídas e fragmentadas. Além das etapas descritas, o autor propõe etapas para a criação e execução de subconsultas e a composição dos resultados. De acordo com o autor, "essa metodologia pode ser aplicada tanto em um banco de dados que permita a fragmentação de bases XML, quanto em um sistema que proporcione uma visão integrada de bancos de dados XML semi-autônomos homogêneos (pois a metodologia não contempla esquemas heterogêneos)". A representação algébrica da consulta é dada por uma expressão da álgebra TLC [34] e, como os fragmentos das bases de dados são também definidos por operações desta álgebra, a localização dos dados pode ser realizada de forma correta e automática.

Neste capítulo, propomos uma metodologia para o processamento de consultas sobre VMs homogêneas, adaptada das metodologias de [19] e [32]. Nossa metodologia consiste de três etapas: (i) *Localização dos dados*, onde são identificadas as VLEs relevantes de acordo com os predicados de projeção e seleção (filtros) da consulta; (ii) *Decomposição da consulta* em subconsultas sobre as VLEs relevantes. Nesta etapa é gerada a expressão global de execução da consulta, uma expressão algébrica composta por operações de união e junção das subconsultas; e (iii) *Otimização global*, onde é gerado o plano de execução da consulta, que consiste de uma estratégia de execução mais eficiente que a simples implementação da expressão global. Para isto, são adotadas heurísticas que objetivam melhorar o desempenho do plano de execução gerado.

É importante observar que não podemos utilizar a metodologia proposta em [19] neste trabalho pois, mesmo trazendo o processamento de consultas para o contexto de bases de dados homogêneas (as VLEs geradas sobre bases heterogêneas representam fragmentos homogêneos da VM), não trabalhamos com a integração de bases formalmente fragmentadas e consideramos que as bases podem conter interseção de dados. Assim, as VLEs geradas (fragmentos) podem não ser disjuntas, condição necessária para a proposta de [19] (regra de correção de disjunção dos fragmentos). Neste trabalho, a integração de dados que estão em interseção é tratada diretamente pela operação de junção, que realiza a fusão (*merge*) de propriedades complexas (incluindo uma estratégia básica de resolução de conflitos), conforme discutido no Capítulo IV.

A seguir, são apresentados maiores detalhes da metodologia proposta. No restante deste capítulo, considere que o termo "VM" refere-se sempre a uma "VM homogênea", definida em termos de VLEs, conforme a arquitetura proposta de três níveis de esquemas.

6.2. Metodologia para o Processamento de Consultas

Neste trabalho, consideramos que uma consulta é sempre realizada sobre uma única visão de mediação (única coleção global) e o elemento primário/subelementos da resposta da consulta (cláusula *return* da consulta XQuery) têm a mesma estrutura do elemento primário/subelementos da VM. Portanto, não há reestruturação dos elementos da VM que foram selecionados pela consulta. Por não ser foco deste trabalho o desenvolvimento de *parsers* para a análise sintática e semântica destas consultas, consideramos que as mesmas são recebidas já validadas, de acordo com as características descritas. Para isto, pode-se, por exemplo, utilizar uma interface gráfica para gerar consultas válidas.

Assim, a etapa de decomposição da consulta da forma proposta por [19] e [32] (transformação em consulta algébrica sobre coleções globais e análise sintática e semântica) não é adotada por nossa metodologia, podendo ser incluída em trabalhos futuros. Está também fora do escopo deste trabalho uma etapa de otimização das subconsultas enviadas para execução nos sites remotos (otimização local), ficando a critério dos sistemas que armazenam as bases de dados a realização desta otimização.

Na metodologia proposta, quando uma consulta sobre uma VM é recebida: (i) são identificadas as VLEs que possuem dados relevantes para a consulta (*localização dos dados*); (ii) a consulta sobre a VM é reescrita em subconsultas sobre as VLEs relevantes (*decomposição*); e (iii) é definida uma ordem de execução das subconsultas e de composição dos resultados que evita o uso desnecessário de operações de maior custo computacional e reduz o fluxo de dados entre as operações (*otimização global*).

Estes passos definem as três etapas da metodologia para o processamento de consultas sobre VMs homogêneas, as quais serão detalhadas a seguir. O algoritmo que implementa esta metodologia, chamado de **ProcessaConsulta**, é apresentado no Apêndice I, juntamente com os demais algoritmos citados neste capítulo.

6.2.1. Localização dos Dados

Seja V_M uma visão de mediação e Q_M uma consulta sobre V_M. Na etapa de localização dos dados, são identificadas as VLEs que possuem informações relevantes para a resposta de Q_M, de acordo com os predicados de projeção e seleção da consulta. O objetivo é eliminar VLEs irrelevantes (redução de fragmentos irrelevantes) para diminuir o volume de dados consultados e melhorar o desempenho da consulta.

O resultado desta etapa consiste na geração do *Grafo das VLEs Relevantes* para Q_M, que contém apenas as VLEs que possuem informações relevantes para a consulta. O grafo de VLEs relevantes é gerado a partir do grafo da hierarquia de VLEs da VM consultada (vide Capítulo V), eliminando-se os nós que representam VLEs que não são relevantes para a consulta. Portanto, o grafo das VLEs relevantes é um subgrafo do grafo da hierarquia de VLEs da VM.

Conforme apresentado no Capítulo V, as VLEs são classificadas em VLEs Primárias e VLEs Secundárias. A geração do grafo das VLEs relevantes é dividida, portanto, em dois passos:

(i) Identificação das VLEs Primárias relevantes;

(ii) Identificação das VLEs Secundárias relevantes;

- **Identificação das VLEs Primárias relevantes**

A identificação das VLEs Primárias relevantes é realizada pelo algoritmo **IdentificaVLEsPrimariasRelevantes**. Neste passo, são eliminados do grafo da hierarquia de VLEs todos os nós (e seus nós filhos) que representam VLEs Primárias que não são relevantes para a resposta da consulta, ou seja, nós que representam VLEs Primárias que não satisfazem os critérios descritos a seguir.

Uma VLE Primária V_{LE} é considerada relevante para uma consulta Q_M sobre uma visão de mediação V_M sse:

1) Considerados os filtros de seleção aplicados em Q_M (os quais formam a expressão de seleção Exp) não existir Assertiva de Correspondência de Global ou de Coleção (ACG) que determine que V_{LE} não possui informações relevantes para Q_M, ou seja, não existir ACG do tipo $[\, V_M[Exp] \cap V_{LE} = \varnothing \,]$;
2) O esquema consolidado de V_{LE} (ver definição 6.1 a seguir) possuir alguma propriedade selecionada por Q_M ou alguma propriedade onde foi aplicado filtro de seleção em Q_M.

O segundo critério é verificado por uma comparação das propriedades selecionadas na consulta (ou onde foram aplicados filtros) com as propriedades do esquema consolidado da VLE. Se alguma propriedade for encontrada no esquema consolidado, o critério é satisfeito. Este critério garante que se uma propriedade selecionada (ou onde foi aplicado filtro) pertencer ao esquema consolidado, então a propriedade pode ser obtida: (i) na própria VLE Primária; ou (ii) nas VLEs Secundárias

que possuem ligação com a VLE Primária. Assim, estando o primeiro critério também satisfeito, a VLE Primária é considerada relevante para a resposta da consulta.

Definição 6.1: Esquema Consolidado de uma VLE Primária

O esquema consolidado de uma VLE Primária consiste da fusão do esquema da VLE Primária com os esquemas de todas as VLE Secundárias referenciadas por ligações da VLE Primária. A fusão dos esquemas é realizada no nível dos elementos envolvidos nas ligações.

O *esquema consolidado de uma VLE Primária* é armazenado no Catálogo do Mediador no momento de geração das VLEs. Seja, por exemplo, a VLE Primária Paciente_1 e as VLEs Secundárias Patologia e Medico, cujos esquemas são exibidos na Figura 6.1(a). Sejam ainda as ligações ℓ_1 entre Paciente_1 e Patologia, e ℓ_2 entre Paciente_1 e Medico. Na Figura 6.1(b) é exibido o esquema consolidado de Paciente_1.

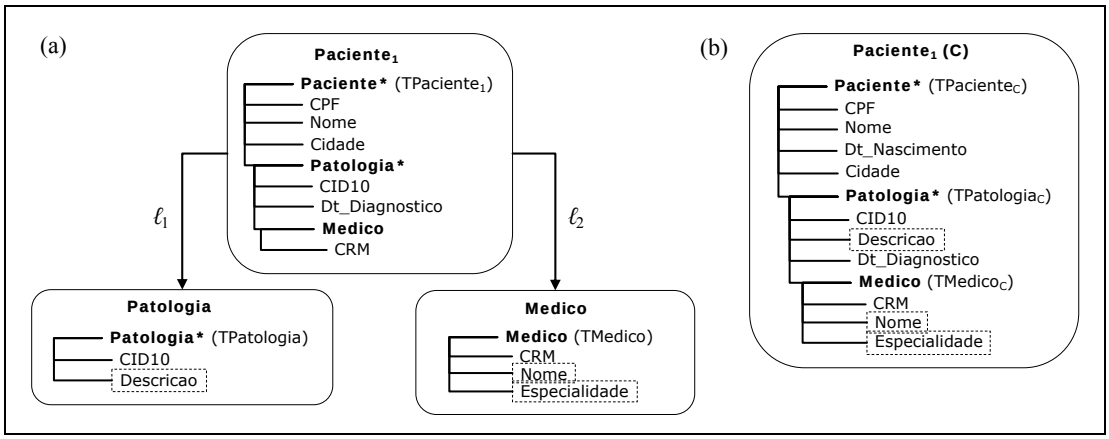


Figura 6.1 – (a) VLE Primária Paciente_1 e VLEs Secundárias Patologia e Medico; (b) Esquema consolidado de Paciente_1

- **Identificação das VLEs Secundárias relevantes**

A identificação das VLEs Secundárias relevantes é realizada pelo algoritmo **IdentificaVLEsSecundariasRelevantes**. Neste passo, são eliminados do grafo da hierarquia de VLEs todos os nós que representam VLEs Secundárias irrelevantes.

Assim, após a remoção dos nós que representam VLEs Primárias não relevantes, os nós que permanecem no grafo são visitados para se identificar as VLEs Secundárias relevantes. As visitas são sempre iniciadas pelos nós que representam as VLEs Primárias relevantes, os quais permaneceram no grafo. Em cada visita é realizada uma comparação entre as propriedades selecionadas por Q_M (e que foram aplicados filtros de seleção em Q_M) e as propriedades da VLE do nó visitado. O objetivo da comparação é identificar a existência de propriedades relevantes na VLE.

Após cada comparação, caso uma propriedade complexa selecionada em Q_M não seja totalmente encontrada na VLE, as subpropriedades não encontradas são procuradas nos nós filhos, que representam VLEs Secundárias. As comparações são realizadas em um processo recursivo até que: (i) não restem propriedades selecionadas por Q_M que não foram encontradas; ou (ii) não existam nós filhos a serem visitados.

Neste trabalho, para que as comparações possam ser realizadas, é gerado um tipo XML T_{Q_M} que representa a consulta Q_M. O tipo T_{Q_M} é gerado trivialmente, onde as propriedades selecionadas (ou que foram aplicados filtros) são mapeadas em elementos de T_{Q_M} (mantendo a estrutura das propriedades complexas em elementos complexos) e os filtros de Q_M são mapeados em atributos dos elementos de T_{Q_M}.

As comparações, portanto, são sempre realizadas entre o tipo T_{Q_M} (ou o tipo de elementos complexos de T_{Q_M}) e o tipo das VLEs. Ainda neste passo, quando as comparações são realizadas:

(i) Caso a VLE possua alguma propriedade em que foram aplicados filtros de seleção em Q_M, o nó que representa a VLE e todos os seus nós ancestrais são assinalados com as chamadas *marcações de filtro* (F). Na geração do plano de execução de Q_M, estas marcações serão utilizadas como um dos critérios para definir a ordem de execução das consultas às VLEs.

(ii) É gerado um tipo XML $T_{Q_{VLE}}$ que representa as propriedades encontradas na VLE do nó visitado. O tipo $T_{Q_{VLE}}$ é chamado de *tipo da consulta local* e é armazenado como atributo do nó visitado. Se a VLE possuir propriedades onde foram aplicados filtros, estes filtros são mapeados em $T_{Q_{VLE}}$.

Ao final do processo, são removidos os nós que representam VLEs Secundárias não relevantes, ou seja, são removidos os nós não visitados e os nós visitados cujas VLEs Secundárias não possuem propriedades relevantes para Q_M. A geração do grafo das VLEs relevantes é concluída.

Seja, por exemplo, o grafo de hierarquia de visões da visão de mediação V_M apresentado na Figura 6.2(a). Suponha que os algoritmos identifiquem que: (i) a VLE Primária V_{P_2} e as VLEs secundárias V_{S_1} e V_{S_2} não são relevantes para o resultado de Q_M; e (ii) a VLE Secundária V_{S_3} possui propriedades em que foram aplicados filtros de seleção em Q_M. O grafo das VLEs relevantes é exibido na Figura 6.2(b). Observe que o nó que representa V_{S_3} e os seus nós ancestrais foram assinalados com as marcações de filtro. Observe ainda que os tipos das consultas locais são adicionados aos nós do grafo.

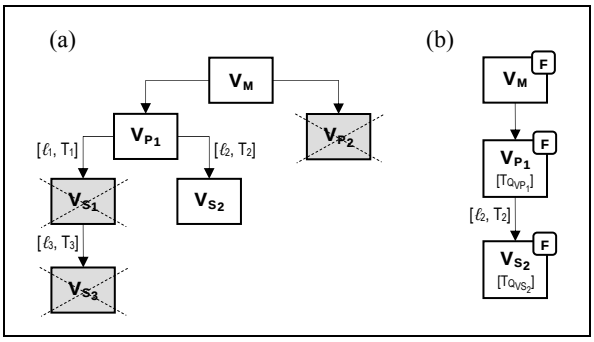


Figura 6.2 – (a) Grafo da Hierarquia de Visões de V_M ; (b) Grafo das VLEs Relevantes para Q_M

6.2.2. Decomposição da Consulta

Nesta etapa, utilizando o grafo das VLEs relevantes, é realizada a decomposição (reescrita) da consulta Q_M em subconsultas sobre as VLEs relevantes e é gerada a *expressão global de execução da consulta*, que consiste de uma expressão algébrica composta por operações de união e junção das subconsultas sobre as VLEs relevantes.

A expressão global é obtida a partir da expressão algébrica (Φ) de reconstrução da VM, removendo-se as VLEs irrelevantes e substituindo-se as VLEs relevantes pelas subconsultas correspondentes. O objetivo, portanto, consiste em reduzir a expressão algébrica de reconstrução da VM em uma expressão algébrica com subconsultas sobre as VLEs relevantes.

A decomposição de Q_M é realizada pelo o algoritmo **DecompoeConsulta**, que também gera a expressão global de execução de Q_M. Em resumo, o algoritmo visita os nós do grafo das VLEs relevantes e, a partir do tipo da consulta local (T_{QVLE}) gerado na etapa anterior, gera a consulta XQuery Q_{VLE} correspondente. A consulta Q_{VLE} é armazenada como atributo do nó visitado no grafo de VLEs relevantes.

Por fim, de forma semelhante à geração da expressão algébrica (Φ) de reconstrução da VM (vide Capítulo **V**), o algoritmo gera a expressão global $\Phi(Q_M)$ de execução da consulta.

Seja, por exemplo, a expressão Φ de reconstrução de V_M da Figura 6.2(a):

$[\Phi: V_M \equiv (((V_{P1} \hat{\mathbf{J}}_{\dots} (V_{S1} \hat{\mathbf{J}}_{\dots} V_{S3})) \hat{\mathbf{J}}_{\dots} V_{S2}) \ddot{\mathbf{U}}_{\dots} V_{P2})]$

Considere o grafo de VLEs relevantes de Q_M da Figura 6.2(b) e que as consultas XQuery geradas a partir dos tipo locais de consulta T_{QVP1} e T_{QVS2} são, respectivamente, Q_{VP1} e Q_{VS2}. A expressão global de execução de Q_M gerada pelo algoritmo é dada por:

$[\Phi: Q_M \equiv (Q_{VP1} \hat{\mathbf{J}}_{\dots} Q_{VS2})]$

6.2.3. Otimização Global

Nesta etapa, é gerado o *plano de execução da consulta*. Em nossa metodologia, não são analisadas diversas estratégias de execução da consulta para a escolha da melhor estratégia, com base em uma função de custo. A geração do plano de execução é suportada por heurísticas que objetivam aumentar a sua eficiência, a saber:

(i) *Redução do custo total das operações de integração*. A ordem de execução e as operações de integração das VLEs Primárias são determinadas com base em características comuns destas VLEs. Assim, são escolhidas operações computacionalmente menos custosas, reduzindo o custo total das operações de integração no processamento de uma consulta.

(ii) *Redução do volume de dados trafegados*. Todas as operações de junção seguem uma estratégia baseada na operação de semi-junção [17], realizando, quando for o caso, uma seleção local para reduzir o volume de dados trafegados entre as bases e o mediador.

(iii) *Execução paralela de operações*. As operações do plano são dispostas de modo que o mediador possa executá-las em paralelo, sempre que possível.

O objetivo desta etapa é, portanto, gerar uma estratégia de execução mais eficiente que a simples implementação da expressão global da consulta. Em trabalhos futuros, poderão ser incluídos mecanismos para se obter estatísticas sobre as bases de dados e o custo das operações, e gerar variações equivalentes do plano de execução para a escolha daquele de melhor desempenho. Na geração destas variações, a seletividade de um predicado em determinada fonte pode ser considerada, por exemplo, fator determinante para a utilização da estratégia baseada em semi-junção.

Neste trabalho, o plano de execução consiste de um *workflow* de processos, os quais dividem-se em: (i) processos responsáveis por enviar subconsultas aos serviços web locais que exportam as VLEs; e (ii) processos responsáveis por integrar os resultados das subconsultas.

Os processos do plano de execução ou, simplesmente, do *workflow de execução da consulta* foram apresentados detalhadamente no Capítulo IV. Estes processos são implementados internamente pelo mediador, que também mantém o controle do fluxo, disparando os processos de consultas aos serviços web locais e os processos de composição da resposta da consulta.

Como alternativa de implementação, o *workflow* de execução pode ser implementado por um processo BPEL [4] de orquestração de serviços web, composto essencialmente por atividades que invocam serviços web locais para executar subconsultas sobre as VLEs e serviços web que implementam as operações de integração dos resultados das subconsultas. Neste caso, o mediador é o responsável por publicar (em tempo de execução da consulta) o processo na máquina BPEL do sistema e enviar uma mensagem (*start*) para o início da sua execução. O resultado do processo BPEL é retornado ao mediador, que o apresenta como resposta da consulta.

O *workflow* de execução da consulta é gerado em três passos:

(i) Geração da Partição de Integração da Consulta – $P_I(Q_M)$;

(ii) Geração do *Workflow* Inicial – $\mathcal{W}_i(Q_M)$;

(iii) Geração do *Workflow* Final – $\mathcal{W}_f(Q_M)$;

A heurística de *Redução do custo total das operações de integração* é adotada na execução dos passos 1 e 2. A heurística de *Execução paralela de operações* é adotada na execução dos passos 2 e 3. A heurística de *Redução do volume de dados trafegados* é adotada na execução do passo 3. A seguir, maiores detalhes destes passos.

- **Geração da Partição de Integração da Consulta – $P_I(Q_M)$**

No *workflow* de execução, a ordem de consulta às VLEs Primárias (e conseqüentemente às VLEs Secundárias que complementam suas informações), a ordem de combinação dos resultados das subconsultas e as operações de integração utilizadas na combinação destes resultados são fatores que podem influenciar consideravelmente a eficiência do processamento de uma consulta.

Intuitivamente, poderíamos considerar que a resposta de uma consulta é obtida pela união simples (*Union*) dos resultados das subconsultas sobre as VLEs Primárias e, em seguida, por junções com os resultados das subconsultas sobre as VLEs Secundárias. Entretanto, como algumas VLEs Primárias podem não ser disjuntas, a união simples poderia resultar em uma resposta com elementos redundantes. Uma segunda alternativa seria realizar a união externa (*Outer Union*) dos resultados obtidos das VLEs Primárias e, em seguida, realizar as junções com os resultados obtidos das VLEs Secundárias. Entretanto, como a operação de *Outer Union* é computacionalmente mais custosa que a operação de *Union* e, em alguns casos, pode ser evitada, esta segunda alternativa é considerada ineficiente.

Neste trabalho, é proposta a seguinte estratégia para a integração das informações exportadas pelas VLEs Primárias (heurística de *redução do custo total das operações de integração*): ao contrário de outros enfoques [49], não realizamos apenas *Union* ou *Outer Union* dos resultados obtidos das VLEs Primárias, mas utilizamos características comuns destas VLEs para gerar um *workflow* que utilize a operação mais adequada quando possível. Para isto, é proposta a criação da partição de integração da consulta, que será utilizada na geração do *workflow* de execução.

A *partição de integração de uma consulta* Q_M ($P_I(Q_M)$) consiste de uma partição do conjunto de VLEs Primárias relevantes em subconjuntos cujas VLEs apresentam características comuns e que podem ser utilizadas para aumentar a eficiência do *workflow* de execução. A partição será utilizada para: (i) priorizar a ordem de consulta às VLEs Primárias, sendo possível também determinar quais consultas podem ser executadas em paralelo; e (ii) identificar o operador mais adequado à integração das VLEs de um mesmo subconjunto ou de subconjuntos diferentes da partição.

A partição de integração de Q_M é gerada neste passo pelo algoritmo **GeraParticaoIntegracaoConsulta**. No final desta seção é apresentada a definição formal da partição de integração de uma consulta.

Em resumo, a partição é dada por $\mathbf{P_I(Q_M) = \{P_{[\equiv]}, P_{[Exp]}, P_{[\neg Exp]}\}}$, onde as VLEs Primárias relevantes são alocadas nos subconjuntos de $P_I(Q_M)$ de acordo com:

(i) As assertivas de correspondência global ou de coleção (ACGs) entre a VM e as VLEs Primárias relevantes;

(ii) Os filtros de seleção que são aplicados sobre as propriedades em Q_M;

(iii) A existência de interseção entre as VLEs Primárias relevantes.

Considere Exp uma expressão de seleção com os filtros aplicados sobre propriedades em Q_M. A expressão de seleção Exp é da forma $[(\delta_1\ \theta_1\ 'v_1') * ... * (\delta_i\ \theta_i\ 'v_i')]$, onde δ_1, ..., δ_i são caminhos do tipo T_{Q_M} da consulta Q_M, θ_1 ,..., θ_i são operadores de comparação, $*$ são operadores lógicos e v_1, ..., v_i são valores escalares. A seguir, veremos como é feita a alocação das VLEs nos subconjuntos da partição.

No subconjunto $P_{[\equiv]}$ são alocadas as VLEs Primárias relevantes que possuem ACGs do tipo $[\psi:V_M[Exp] \equiv V_{LE_{[\equiv]}}]$; ou do tipo $[\psi:V_M \equiv V_{LE_{[\equiv]}}]$ cujo esquema consolidado possui todas as propriedades sobre as quais os filtros de Exp foram aplicados.

No subconjunto $P_{[Exp]}$ são alocadas as VLEs Primárias relevantes que não se enquadram no subconjunto $P_{[\equiv]}$ e o seu esquema consolidado possui pelo menos uma

propriedade sobre a qual um filtro de Exp (por exemplo (δ_i θ_i 'v_i')) foi aplicado.

Por último, no subconjunto $P_{[\neg Exp]}$ são alocadas as VLEs Primárias relevantes que não se enquadram nos subconjuntos anteriores e cujo esquema consolidado não possui qualquer propriedade sobre a qual um filtro de Exp foi aplicado.

Veremos adiante que a execução das subconsultas é sempre iniciada pelas VLEs dos subconjuntos $P_{[\equiv]}$ ou $P_{[Exp]}$ (ou pelas VLEs Secundárias ligadas a estas VLEs). Isto ocorre porque apenas as VLEs destes subconjuntos podem responder aos filtros de Exp.

Os subconjuntos $P_{[Exp]}$ e $P_{[\neg Exp]}$ subdividem-se em novos subconjuntos, de acordo com a existência de restrições de interseção entre as VLEs Primárias. Conforme vimos no Capítulo V, estas restrições de interseção são geradas a partir das restrições de interseção entre as coleções-base sobre as quais as VLEs Primárias estão definidas. Desta forma, em subconjuntos $P_{[Exp]_i}$ ou $P_{[\neg Exp]_j}$ estão alocadas VLEs Primárias relevantes que possuem interseção não-vazia. Se duas VLEs estão alocadas em subconjuntos $P_{[Exp]_i}$ ou $P_{[\neg Exp]_j}$ distintos, então essas VLEs possuem interseção vazia (disjuntas), pois, caso contrário, estariam no mesmo subconjunto.

Essa subdivisão é utilizada para gerar um *workflow* de execução com operações de integração menos custosas, quando for o caso. Assim, entre coleções obtidas de VLEs de um mesmo subconjunto $P_{[Exp]_i}$ ou $P_{[\neg Exp]_j}$ é sempre realizada a *Outer Union* (Ü), pois podem existir informações de uma mesma entidade nas duas coleções. Já entre coleções obtidas de VLEs de subconjuntos diferentes, por possuírem interseção vazia (disjuntas), é realizada apenas a *Union* (U), que é uma operação de menor custo computacional.

É importante destacar a relevância do subconjunto $P_{[\neg Exp]}$. Durante o processamento da consulta, mesmo as VLEs deste subconjunto não possuindo propriedades onde foram aplicados os filtros de Exp, ou seja, mesmo não conseguindo responder aos filtros da consulta, elas são consultadas com o objetivo de encontrar informações complementares para os resultados obtidos das VLEs dos demais subconjuntos, que conseguem responder aos filtros. Portanto, as VLEs do subconjunto $P_{[\neg Exp]}$ não podem ser descartadas.

A seguir, apresentamos a definição formal da partição de integração de uma consulta Q_M. No restante desta seção, considere Q_M uma consulta sobre uma visão de mediação V_M, Exp = [(δ_1 θ_1 'v_1') ∗ ... ∗ (δ_i θ_i 'v_i')] a expressão de seleção com os filtros aplicados sobre propriedades em Q_M e V_{LE} uma VLE Primária de V_M.

Definição 6.2: Expressão de Seleção Suportada por uma VLE

Seja V_{LE}^C o esquema consolidado de V_{LE}. Temos que:

- V_{LE} *suporta totalmente* Exp sse $\forall\ \delta \in \{\delta_1, ..., \delta_i\}$, δ é um caminho de V_{LE}^C;

- V_{LE} *suporta parcialmente* Exp sse $\exists\ \delta_p \in \{\delta_1, ..., \delta_i\}$ e $\exists\ \delta_q \in \{\delta_1, ..., \delta_i\}$ tal que δ_p é um caminho de V_{LE}^C e δ_q não é um caminho de V_{LE}^C. □

Definição 6.3: Partição de Integração de uma Consulta Q_M

Seja VLESp o conjunto de VLEs Primárias de V_M que são relevantes para Q_M. Seja ainda ACGs o conjunto de assertivas de correspondência de coleção (conjunto ψ) entre V_M e suas VLEs.

O conjunto VLESp' é dado como se segue:

$$VLESp' = \{ V_{LE} \mid V_{LE} \in V_{LEs} ; \neg\exists\ \psi \in ACGs \text{ tal que } (\psi{:}V_M[Exp] \cap V_{LE} \equiv \varnothing) \}$$

A partição de integração de Q_M, denotada por $P_I(Q_M) = \{P_{[\equiv]}, P_{[Exp]}, P_{[\neg Exp]}\}$, é dada por:

- $VLESp' = P_{[\equiv]} \cup P_{[Exp]} \cup P_{[\neg Exp]}$

- $P_{[\equiv]} = \{ V_{LE_{[\equiv]}} \mid V_{LE_{[\equiv]}} \in VLESp' ; \exists\ \psi \in ACGs \text{ tal que } (\psi{:}V_M[Exp] \equiv V_{LE_{[\equiv]}}) \text{ ou } (\psi{:}V_M \equiv V_{LE_{[\equiv]}} \text{ e } V_{LE_{[\equiv]}} \text{ suporta completamente Exp}) \};$

- $P_{[Exp]} = \{P_{[Exp]_i} \mid i \geq 0\}$, onde:

  - $P_{[Exp]_i} = \{ V_{LE_{[Exp]_i}} \mid V_{LE_{[Exp]_i}} \in VLESp' ; V_{LE_{[Exp]_i}} \notin P_{[\equiv]} ; V_{LE_{[Exp]_i}} \text{ suporta parcialmente ou completamente Exp} ; \forall\ V_{LE_{[Exp]_j}} \in P_{[Exp]_i} , V_{LE_{[Exp]_i}} \cap V_{LE_{[Exp]_j}} \neq \varnothing\};$

  - $\forall\ P_{[Exp]_i} \in P_{[Exp]}$ e $\forall\ P_{[Exp]_j} \in P_{[Exp]}, P_{[Exp]_i} \cap P_{[Exp]_j} = \varnothing$.

- $P_{[\neg Exp]} = \{P_{[\neg Exp]_i} \mid i \geq 0\}$, onde:

  - $P_{[\neg Exp]_i} = \{ V_{LE_{[\neg Exp]_i}} \mid V_{LE_{[\neg Exp]_i}} \in VLESp' ; V_{LE_{[\neg Exp]_i}} \notin P_{[\equiv]} ; V_{LE_{[\neg Exp]_i}} \text{ não suporta Exp} ; \forall\ V_{LE_{[\neg Exp]_j}} \in P_{[\neg Exp]_i} , V_{LE_{[\neg Exp]_i}} \cap V_{LE_{[\neg Exp]_j}} \neq \varnothing\};$

  - $\forall\ P_{[\neg Exp]_i} \in P_{[\neg Exp]}$ e $\forall\ P_{[\neg Exp]_j} \in P_{[\neg Exp]}, P_{[\neg Exp]_i} \cap P_{[\neg Exp]_j} = \varnothing$. □

- **Geração do *Workflow* Inicial – $\mathcal{W}_i(Q_M)$**

Neste passo, a partir da partição de integração de Q_M, é realizada a geração do *workflow* inicial $\mathcal{W}_i(Q_M)$. O *workflow* $\mathcal{W}_i(Q_M)$ consiste de uma seqüência de processos responsáveis por consultar as VLEs Primárias relevantes e processos que realizam operações de união, união externa ou junção das coleções obtidas nestas subconsultas, de modo que não sejam utilizadas operações de maior custo computacional quando não for necessário.

O algoritmo responsável por este passo é o **GeraWorkflowExecucaoInicial**. As regras 6.1 e 6.2, descritas a seguir, são aplicadas no início da geração do *workflow*.

Definição 6.4: Expressão de Seleção Suportada Totalmente por $P_{[Exp]_i}$

Seja $P_I(Q_M) = \{P_{[\equiv]}, P_{[Exp]}, P_{[\neg Exp]}\}$ a partição de integração de Q_M, de acordo com a Definição 6.3, e $P_{[Exp]_i}$ um subconjunto qualquer de $P_{[Exp]}$. O subconjunto $P_{[Exp]_i}$ *suporta totalmente* Exp sse:

$\forall \delta \in \{\delta_1, ..., \delta_j\}, \exists V_{LE_{[Exp]_i}} \in P_{[Exp]_i}$ tal que δ é um caminho de $V_{LE_{[Exp]_i}}{}^C$, onde $V_{LE_{[Exp]_i}}{}^C$ é o esquema consolidado de $V_{LE_{[Exp]_i}}$. □

Regra 6.1: Condição Essencial para a Geração do *Workflow* $\mathcal{W}_i(Q_M)$

Se o usuário definiu uma consulta onde a expressão de seleção Exp não é suportada totalmente por algum subconjunto de $P_{[Exp]}$ e $P_{[\equiv]}$ é vazio, então o workflow $\mathcal{W}_i$ não pode ser gerado e a resposta de Q_M será vazia. Mais formalmente:

Seja $P_I(Q_M) = \{P_{[\equiv]}, P_{[Exp]}, P_{[\neg Exp]}\}$ a partição de integração de Q_M e $P_{[Exp]}'$ o conjunto dado por:

$P_{[Exp]}' = \{ P_{[Exp]}'_i \mid P_{[Exp]}'_i \in P_{[Exp]}$ e $P_{[Exp]}'_i$ suporta totalmente Exp $\}$

Se $P_{[\equiv]} = \varnothing$ e $P_{[Exp]}' = \varnothing$ então o *workflow* $\mathcal{W}_i(Q_M)$ não pode ser gerado e a resposta de Q_M é vazia.

A regra 6.1 evita que o *workflow* $\mathcal{W}_i(Q_M)$ seja gerado sem necessidade, pois nunca seria possível garantir que as entidades retornadas satisfazem a expressão de seleção da consulta Q_M, ou seja: (i) não existem VLEs na partição $P_{[\equiv]}$; e (ii) não existem VLEs na partição $P_{[exp]}$ que estejam em interseção e, juntas, consigam responder aos filtros da expressão de seleção.

Regra 6.2: Exclusão de subconjuntos $P_{[Exp]_i}$ irrelevantes para Q_M em $P_I(Q_M)$

Se o subconjunto $P_{[\equiv]}$ é vazio, então subconjuntos $P_{[Exp]_i}$ que não suportam totalmente a expressão de seleção Exp devem ser excluídos de $P_I(Q_M)$.

Mais formalmente, seja $P_I(Q_M) = \{P_{[\equiv]}, P_{[Exp]}, P_{[\neg Exp]}\}$ a partição de integração de Q_M e $P_{[Exp]}'$ o conjunto dado por:

$P_{[Exp]}' = \{ P_{[Exp]}'_i \mid P_{[Exp]}'_i \in P_{[Exp]}$ e $P_{[Exp]}'_i$ suporta totalmente Exp $\}$

Se $P_{[\equiv]} = \varnothing$ e $P_{[Exp]} \neq \varnothing$ então o conjunto $P_{[Exp]}$ de $P_I(Q_M)$ é trocado por $P_{[Exp]}'$, ou seja, $P_{[Exp]} := P_{[Exp]}'$.

Caso a Regra 6.1 seja satisfeita, a Regra 6.2 é aplicada. Os subconjuntos $P_{[Exp]_i}$ são removidos, pois uma vez que não suportam totalmente Exp e as VLEs que os compõem possuem interseção vazia com os demais subconjuntos de $P_{[Exp]}$, não será possível obter destes subconjuntos informações relevantes para o resultado da consulta.

Após a aplicação das regras, o *workflow* inicial $\mathcal{W}_i(Q_M)$ é gerado. De acordo com o conteúdo do conjunto $P_{[\equiv]}$, o *workflow* $\mathcal{W}_i(Q_M)$ pode ter duas configurações, conforme será exibido a seguir:

Caso 1: $P_{[\equiv]} \neq \varnothing$

Seja $P_i(Q_M) = \{P_{[\equiv]}, P_{[Exp]}, P_{[\neg Exp]}\}$ a partição de integração de Q_M após a aplicação da regra 6.2 , conforme se segue:

- $P_{[\equiv]} = \{ V_{LE[\equiv]_1} , V_{LE[\equiv]_2} , ..., V_{LE[\equiv]_k} \}$, para $k \geq 0$;
- $P_{[Exp]} = \{ P_{[Exp]_1} , P_{[Exp]_2} , ..., P_{[Exp]_i} \}$, para $i \geq 0$, onde:
  - $P_{[Exp]_1} = \{ V_{LE[Exp]_{11}} , V_{LE[Exp]_{12}} , ..., V_{LE[Exp]_{1m_1}} \}$, para $m_1 \geq 0$
  - $P_{[Exp]_2} = \{ V_{LE[Exp]_{21}} , V_{LE[Exp]_{22}} , ..., V_{LE[Exp]_{2m_2}} \}$, para $m_2 \geq 0$
  - . . .
  - $P_{[Exp]_i} = \{ V_{LE[Exp]_{i1}} , V_{LE[Exp]_{i2}} , ..., V_{LE[Exp]_{im_i}} \}$, para $m_i \geq 0$
- $P_{[\neg Exp]} = \{ P_{[\neg Exp]_1} , P_{[\neg Exp]_2} , ..., P_{[\neg Exp]_j} \}$, para $j \geq 0$, onde:
  - $P_{[\neg Exp]_1} = \{ V_{LE[\neg Exp]_{11}} , V_{LE[\neg Exp]_{12}} , ..., V_{LE[\neg Exp]_{1n_1}} \}$, para $n_1 \geq 0$
  - $P_{[\neg Exp]_2} = \{ V_{LE[\neg Exp]_{21}} , V_{LE[\neg Exp]_{22}} , ..., V_{LE[\neg Exp]_{2n_2}} \}$, para $n_2 \geq 0$
  - . . .
  - $P_{[\neg Exp]_j} = \{ V_{LE[\neg Exp]_{j1}} , V_{LE[\neg Exp]_{j2}} , ..., V_{LE[\neg Exp]_{jn_j}} \}$, para $n_j \geq 0$

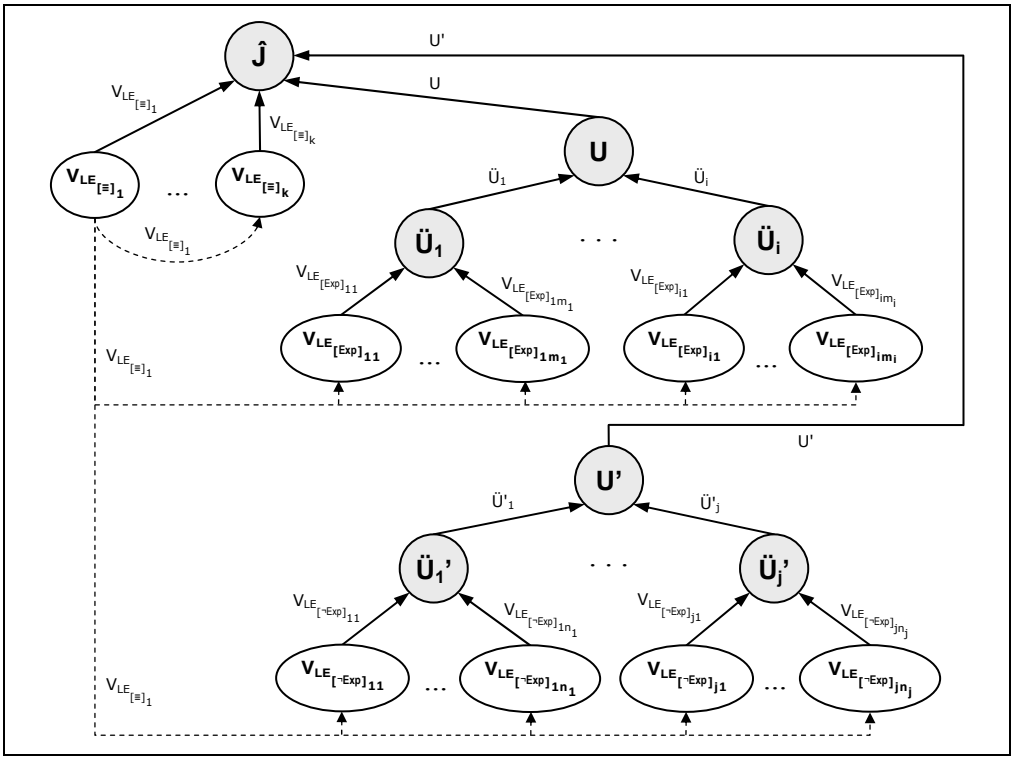


Figura 6.3 – Workflow $\mathcal{W}_i(Q_M)$ (Caso 1)

O *workflow* $\mathcal{W}_i(Q_M)$ (caso 1) é mostrado na Figura 6.3. Para simplificação da figura, os Processos-Consulta são representados apenas pelo nome da VLE, os Processos-Operadores são representados apenas pelos nomes das operações de integração e os atributos de todos os processos não são exibidos.

Caso 2: $P_{[\equiv]} = \varnothing$

Seja $P_I(Q_M) = \{P_{[\equiv]}, P_{[Exp]}, P_{[\neg Exp]}\}$ a partição de integração de Q_M, mostrada no caso anterior. Entretanto, considere que o conjunto $P_{[\equiv]}$ é vazio. Para este caso, o *workflow* $\mathcal{W}_i(Q_M)$ (caso 2) é mostrado na Figura 6.4.

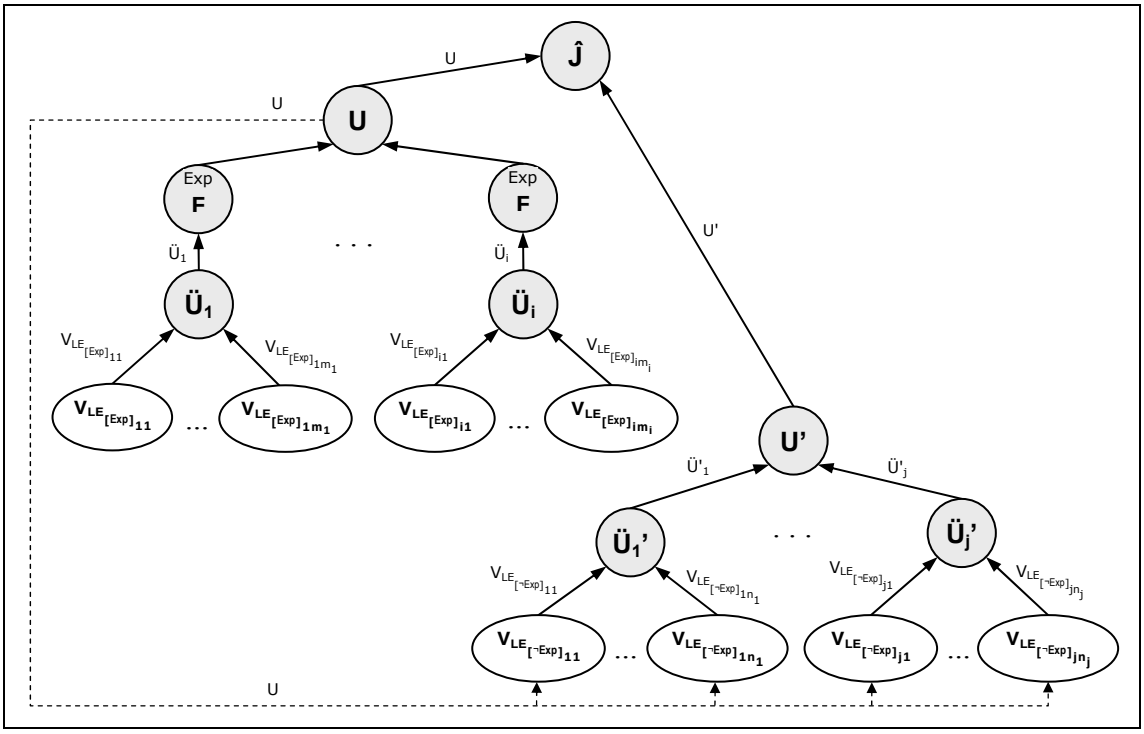


Figura 6.4 – Workflow $\mathcal{W}_i(Q_M)$ (Caso 2)

Considerando a ordem de execução dos processos de um *workflow* (vide Seção 4.4.2), observe que as subconsultas às VLEs Primárias são sempre iniciadas por aquelas que podem responder aos filtros da expressão de seleção Exp. No Caso 1, as subconsultas são iniciadas pelas VLEs do subconjunto $P_{[\equiv]}$ e, no Caso 2, pelas VLEs dos subconjuntos $P_{[Exp]_i}$ que suportam totalmente Exp. Desta forma, as VLEs dos demais subconjuntos são consultadas somente após terem sido obtidas as entidades que satisfazem Exp. Observe ainda que todas as coleções que são obtidas de VLEs de um mesmo subconjunto $P_{[Exp]_i}$ ou $P_{[\neg Exp]_j}$ são integradas por *Outer Union* (Ü), pois possuem interseção não-vazia. Já as coleções obtidas de VLEs de subconjuntos distintos $P_{[Exp]_i}$ ou $P_{[\neg Exp]_j}$ são integradas por *Union* (U), uma vez que representam coleções disjuntas.

- **Geração do *Workflow* Final – $\mathcal{W}_f(Q_M)$**

Neste passo, utilizando o grafo das VLEs relevantes, são adicionados ao *workflow* inicial $\mathcal{W}_i(Q_M)$ os processos responsáveis por consultar e integrar informações das VLEs Secundárias relevantes, resultando no *workflow* final $\mathcal{W}_f(Q_M)$. Paralelamente, as subconsultas sobre as VLEs relevantes são atribuídas aos Processos-Consulta, responsáveis por ser enviá-las aos serviços web locais.

O algoritmo responsável pela geração do *workflow* final $\mathcal{W}_f(Q_M)$ é o **GeraWorkflowExecucaoFinal**. Em resumo, o algoritmo visita os nós do grafo que representa o *workflow* $\mathcal{W}_i$ e, para cada nó N que representa um Processo-Consulta a uma VLE Primária V_{LEP}, verifica no grafo das VLEs relevantes se existem VLEs Secundárias hierarquicamente ligadas à V_{LEP}. Em caso positivo, adiciona os processos responsáveis por consultar e integrar as informações destas VLEs Secundárias, conforme se segue:

1) O nó N é substituído por um nó J, que representa um Processo-Junção Múltipla (PJM);
2) O nó N é adicionado como filho do nó J e representa o Processo-Consulta que fornece a coleção base das operações de junção em J;
3) Os nós que representam VLEs Secundárias ligadas à V_{LEP} são consultados no grafo das VLEs relevantes:

3.1) Se o nó possuir marcação de filtro e o nó N não possuir nó filho (nó que fornece valores para algum parâmetro do nó N), é adicionado no *workflow* $\mathcal{W}_i$ um nó S como filho do nó J, à esquerda do nó N, representando um Processo-Consulta à VLE Secundária. O nó S também é adicionado como filho do nó N, representando o Processo-Consulta que fornece um conjunto de valores para os filtros da subconsulta do nó N (estratégia de semi-junção);

3.2) Caso contrário, é adicionado no *workflow* $\mathcal{W}_i$ um nó S como filho do nó J, à direita do nó N, representando um Processo-Consulta à VLE Secundária. O nó N também é adicionado como filho do nó S, representando o Processo-Consulta que fornece um conjunto de valores para os filtros da subconsulta do nó S (estratégia baseada na operação de semi-junção);

3.3) Se o nó possuir outros nós-filhos, o algoritmo realiza recursivamente a substituição do nó adicionado S, a partir do passo 1.

No momento da visita ou da inclusão de Processos-Consulta no *workflow*, são atribuídas a estes processos as respectivas subconsultas, geradas da fase de decomposição. Para seguir a heurística de redução do volume de dados trafegados, onde todas as operações de junção são realizadas seguindo uma estratégia baseada na operação de semi-junção, o algoritmo adiciona a estas consultas, quando for o caso, uma cláusula *where* que filtra o seu resultado de acordo com um conjunto de valores (projeção do atributo de junção) recebidos de um outro Processo-Consulta (casos dos passos 3.1 e 3.2, descritos acima).

Uma característica importante deste algoritmo é que, no caso de Processos-Consulta cujos resultados serão integrados por uma operação de junção, as propriedades selecionadas pela subconsulta de um Processo-Consulta são removidas pelo algoritmo (exceção para as chaves-primárias) das propriedades selecionadas pelas subconsultas dos demais Processos-Consulta. Assim, se o algoritmo identifica que a consulta local que será atribuída ao Processo-Consulta seleciona apenas propriedades chaves-primárias e não aplica filtros sobre estas propriedades, o Processo-Consulta é eliminado do grafo, pois não estaria buscando propriedades relevantes para a resposta.

Suponha, por exemplo, o *workflow* inicial $\mathcal{W}_i(Q_M)$ exibido na Figura 6.5(a) e o grafo de VLEs relevantes de Q_M exibido na Figura 6.5(b). O *workflow* final $\mathcal{W}_f(Q_M)$, gerado pelo algoritmo **GeraWorkflowResolucaoFinal**, é exibido na Figura 6.5(c).

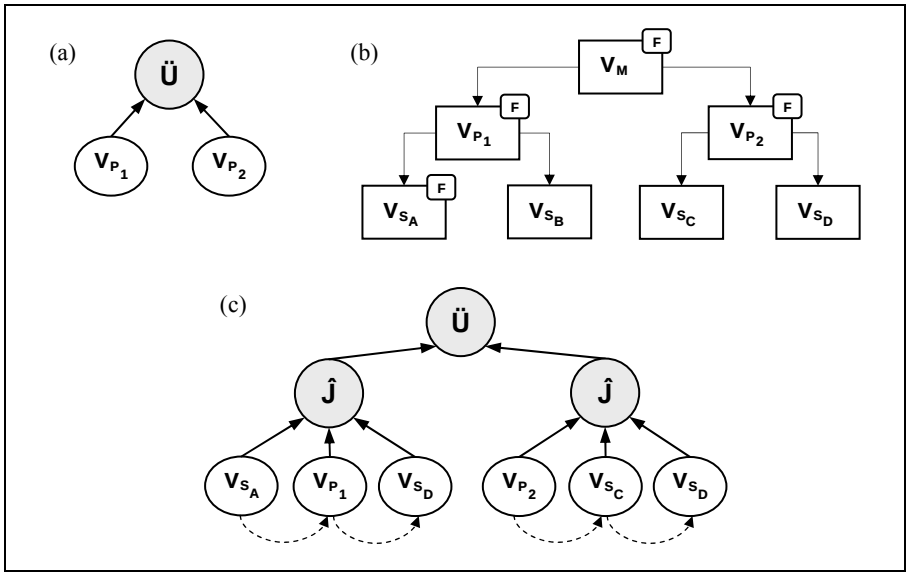


Figura 6.5 – (a) Workflow $\mathcal{W}_i(Q_M)$; (b) Grafo das VLEs relevantes para Q_M ; (c) Workflow $\mathcal{W}_f(Q_M)$

6.3. Exemplo

Nesta seção, é apresentado um exemplo para ilustrar a metodologia de processamento de consultas proposta neste trabalho. Seja a visão de mediação $Paciente_M$, as VLEs Primárias $Paciente_1$ e $Paciente_2$, e as VLEs Secundárias $Patologia_3$ e $Medico_4$ do exemplo do Capítulo III, exibidas na Figura 6.6. Seja $Q_{Paciente}$ uma consulta sobre $Paciente_M$, representada em XQuery na Figura 6.7(a). O tipo XML que representa $Q_{Paciente}$ é exibido na Figura 6.7(b).

Conforme apresentado na Seção 6.2, o processamento de $Q_{Paciente}$ é realizado em três etapas: localização dos dados, decomposição da consulta e otimização global, os quais serão discutidas a seguir.

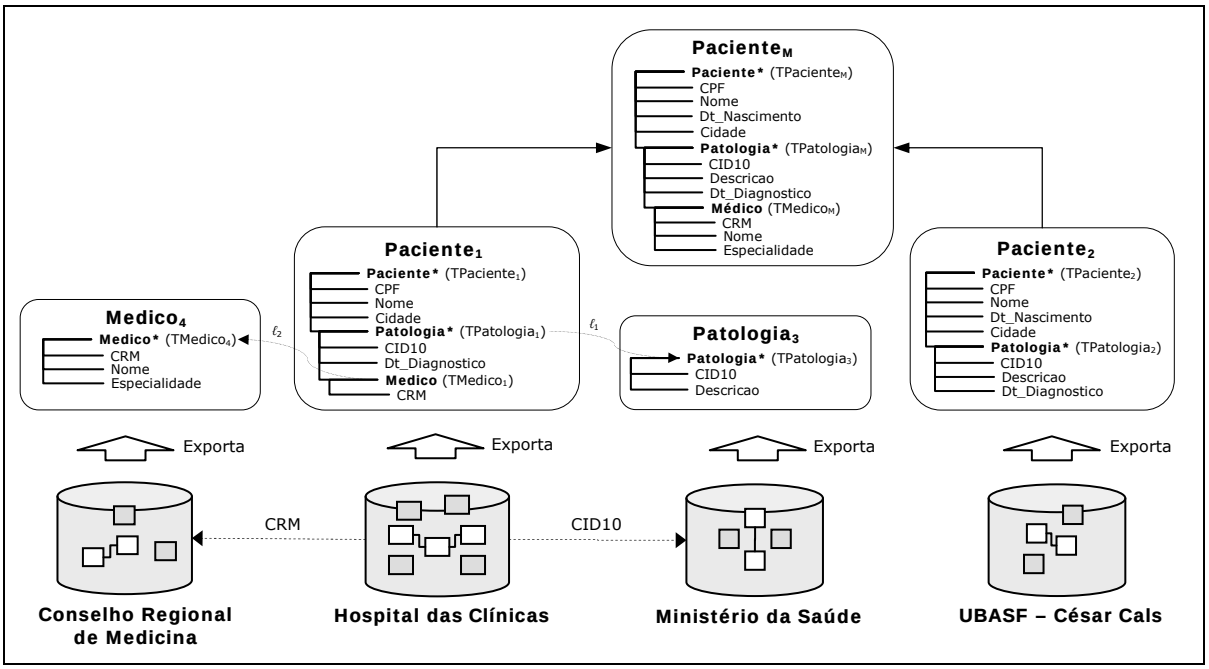


Figura 6.6 – VM $Paciente_M$; VLEs Primárias $Paciente_1$ e $Paciente_2$; e VLEs Secundárias $Patologia_3$ e $Medico_4$

(a)

```
for $x1 in VIEW("PacienteM")/Paciente
where $x1/Cidade = 'Fortaleza'
return <paciente> {
          $x1/CPF,
          $x1/Nome,
          $x1/Dt_Nascimento,
          for $x2 in $x1/Patologia
          return <patologia>{
                    $x2/CID10,
                    $x2/Descricao,
                    $x2/Dt_Diagnostico
                 }</patologia>
       }</paciente>
```

(b)

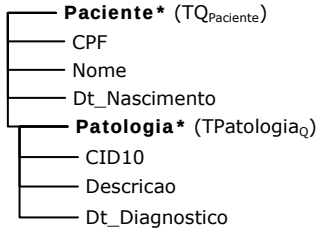


Figura 6.7 – (a) Consulta $Q_{Paciente}$; (b) Tipo XML de $Q_{Paciente}$

6.3.1. Localização dos Dados

- **Identificação das VLEs Primárias relevantes**

Uma VLE Primária V_{LE} é relevante para $Q_{Paciente}$ sse: (i) não existir ACG de $Paciente_M$ do tipo $[Paciente_M[Paciente/Cidade = \text{'Fortaleza'}] \cap V_{LE} = \varnothing]$ e (ii) o esquema consolidado de V_{LE} possuir informações relevantes para a consulta.

A primeira condição é verificada por uma consulta às ACGs de $Paciente_M$. A segunda condição é verificada através de uma comparação das propriedades selecionadas pela consulta $Q_{Paciente}$ (ou as propriedades onde foram aplicados filtros em $Q_{Paciente}$) com as propriedades dos esquemas consolidados de $Paciente_1$ e $Paciente_2$. A partir desta comparação, são identificadas as VLEs Primárias que possuem informações relevantes para $Q_{Paciente}$.

Identificadas as VLEs Primárias relevantes, são eliminados do grafo da hierarquia de VLEs de $Paciente_M$ todos os nós (e seus nós filhos) que representam VLEs Primárias que não são relevantes para a resposta de $Q_{Paciente}$.

Assim, considere o seguinte conjunto de ACGs para $Paciente_M$:

$$ACGs(Paciente_M) = \{\ \psi_1: Paciente_M \supset Paciente_1\ ;$$
$$\psi_2: Paciente_M \supset Paciente_2\ ;$$
$$\psi_3: Paciente_M[Paciente/Cidade='Fortaleza'] \equiv Paciente_1\ \}$$

Observe, então, que considerando o filtro $[Paciente/Cidade = \text{'Fortaleza'}]$, não existem ACGs que excluem $Paciente_1$ e $Paciente_2$ da lista de VLEs Primárias relevantes. Observe ainda que $Q_{Paciente}$ seleciona, por exemplo, as propriedades $Paciente/CPF$ e $Paciente/Nome$, que pertencem tanto ao esquema consolidado de $Paciente_1$ como ao esquema consolidado de $Paciente_2$. Assim, as duas VLEs Primárias possuem informações relevantes para $Q_{Paciente}$.

Portanto, estando satisfeitas as duas condições, as VLEs Primárias $Paciente_1$ e $Paciente_2$ são consideradas VLEs Primárias relevantes para $Q_{Paciente}$ e os nós que representam estas VLEs permanecem no grafo das VLEs relevantes.

- **Identificação das VLEs Secundárias relevantes**

As VLEs Secundárias relevantes para $Q_{Paciente}$ são identificadas através da visita aos nós que permanecem no grafo das VLEs relevantes e da comparação das propriedades selecionadas pela consulta $Q_{Paciente}$ (ou as propriedades onde foram aplicados filtros) com as propriedades das VLEs dos nós visitados.

Ainda neste passo, os nós (e todos os seus nós ancestrais) que representam VLEs que possuem propriedades onde foram aplicados filtros de seleção em $Q_{Paciente}$ são assinalados com as *marcações de filtro* (F) e, após a comparação do tipo da consulta com os tipos das VLEs Primárias e Secundárias, são gerados os *tipos das consulta locais*, armazenados como atributos dos nós do grafo das VLEs relevantes.

Identificadas das VLEs Secundárias relevantes, assinaladas as marcações de filtro e gerados os tipos das consultas locais, são eliminados do grafo da hierarquia de VLEs todos os nós que representam VLEs Secundárias que não são relevantes para a resposta da consulta.

Na Figura 6.8(a) é exibido o grafo de hierarquia das VLEs de $Paciente_M$. Como as VLEs Primárias $Paciente_1$ e $Paciente_2$ são relevantes para $Q_{Paciente}$, os nós que representam estas VLEs foram mantidos no grafo no passo anterior. Na comparação de propriedades de $Q_{Paciente}$ com as propriedades de $Paciente_1$, identificou-se que $Paciente_1$ possui a propriedade Paciente/Cidade, onde foi aplicado o filtro [Paciente/Cidade = 'Fortaleza']. Portanto, o nó que representa $Paciente_1$ e todos os seus nós ancestrais são assinalados com marcações de filtro.

No processo de visita aos nós que representam VLEs Secundárias e na comparação de propriedades, apenas $Patologia_3$ foi identificada como relevante, pois a consulta $Q_{Paciente}$ seleciona a propriedade Patologia/Descricao, que é encontrada apenas em $Patologia_3$. A VLE Secundária $Medico_4$ não possui informações relevantes para $Q_{Paciente}$, pois a consulta não seleciona propriedades de Medico. Portanto, o nó que representa $Medico_4$ é removido do grafo. Nas comparações de propriedades, foram gerados os tipos das consultas locais $TQ_{Paciente_1}$, $TQ_{Patologia_3}$ e $TQ_{Paciente_2}$.

Finalizado o processo de assinalar as marcações de filtro, gerar os tipos das consultas locais e remover os nós que representam VLEs não relevantes, está concluída geração o grafo das VLEs relevantes para $Q_{Paciente}$, exibido na Figura 6.8(b).

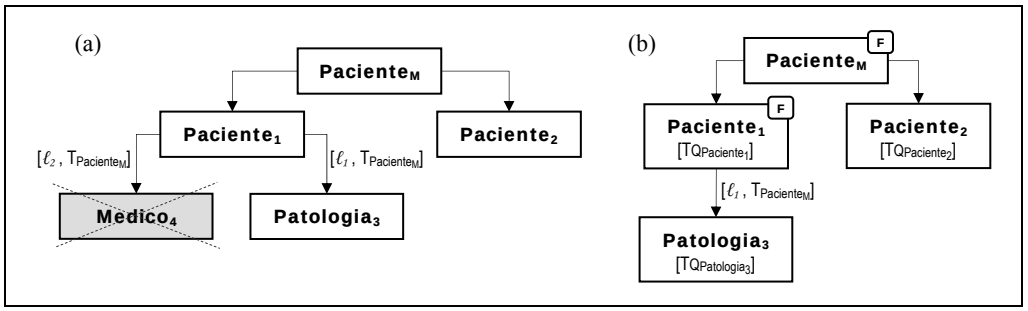


Figura 6.8 – (a) Grafo da hierarquia das VLEs de $Paciente_M$; (b) Grafo das VLEs relevantes para $Q_{Paciente}$

6.3.2. Decomposição da Consulta

Nesta etapa, visitando-se os nós do grafo das VLEs relevantes de $Q_{Paciente}$, é realizada a decomposição de $Q_{Paciente}$ em subconsultas Q_1, Q_2 e Q_3, respectivamente sobre as VLEs relevantes $Paciente_1$, $Paciente_2$ e $Patologia_3$. É também gerada a expressão global de execução de $Q_{Paciente}$, que consiste de uma redução da expressão algébrica de reconstrução de $Paciente_M$, composta por operações de união e junção das subconsultas.

Conforme é mostrado na Figura 6.9, a partir dos tipos das consultas locais ($TQ_{Paciente1}$, $TQ_{Paciente2}$ e $TQ_{Patologia3}$), os quais foram gerados na etapa anterior pelas comparações de tipos, são geradas as subconsultas XQuery Q_1, Q_2 e Q_3. Observe que a expressão de seleção de $Q_{Paciente}$ é mapeada em Q_1, pois $Paciente_1$ possui a propriedade Paciente/Cidade sobre a qual foi aplicado a expressão [Paciente/Cidade = 'Fortaleza'].

Por fim, é gerada a expressão global de execução de $Q_{Paciente}$, substituindo-se as VLEs relevantes pelas subconsultas correspondentes.

Seja, então, a expressão Φ de reconstrução de $Paciente_M$:

$[\Phi: Paciente_M \equiv ((Paciente_1 \hat{J}_{...} Patologia_3) \hat{J}_{...} Medico_4) \ddot{U}_{...} Paciente_2]$

A expressão global de execução de $Q_{Paciente}$ é dada por:

$[\Phi: Q_{Paciente} \equiv (Q_1 \hat{J}_{...} Q_3) \ddot{U}_{...} Q_2]$

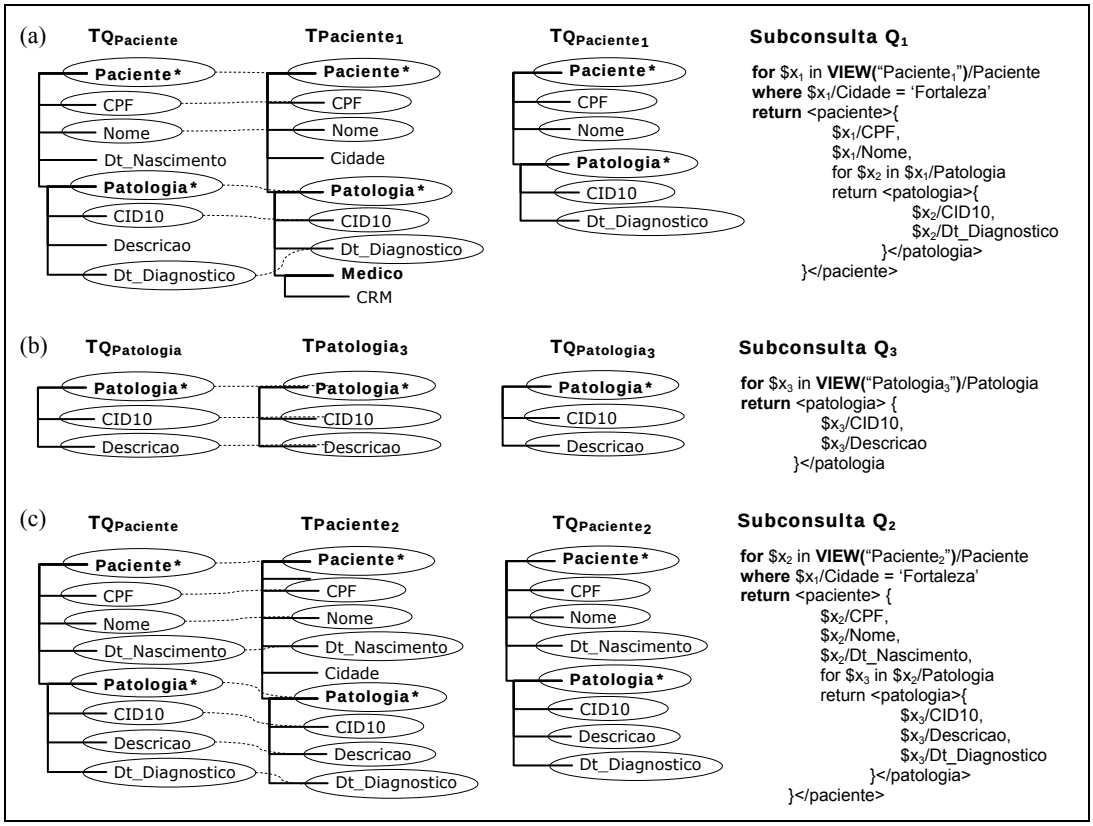


Figura 6.9 – (a)(b)(c) Decomposição de $Q_{Paciente}$ em subconsultas Q_1, Q_3 e Q_2 sobre as VLEs relevantes $Paciente_1$, $Patologia_3$ e $Paciente_2$, respectivamente.

6.3.3. Otimização Global

O plano de execução de $Q_{Paciente}$ é gerado nesta etapa. Conforme visto na Seção 6.2.3, este plano consiste de uma estratégia de execução da consulta mais eficiente que a simples implementação da sua expressão global. O plano é gerado em três passos:

(i) Geração da Partição de Integração de $Q_{Paciente}$ – $P_I(Q_{Paciente})$;

(ii) Geração do *Workflow* Inicial – $\mathcal{W}_i(Q_{Paciente})$;

(iii) Geração do *Workflow* Final – $\mathcal{W}_f(Q_{Paciente})$;

- **Geração da Partição de Integração de $Q_{Paciente}$ – $P_I(Q_{Paciente})$**

Conforme a definição de partição de integração de uma consulta (Definição 6.3), as VLEs são alocadas nos subconjuntos da partição avaliando-se: (i) as ACGs de $Paciente_M$; (ii) a expressão de seleção de $Q_{Paciente}$; e (iii) as restrições de interseção entre as VLEs.

Inicialmente, todas as ACGs de $Paciente_M$ são recuperadas do catálogo do mediador. Em seguida, é realizada uma seleção daquelas ACGs que contemplam os filtros da expressão de seleção de $Q_{Paciente}$, ou seja, ACGs do tipo $[\psi_i: V_M[Exp] \equiv V_{LE}]$. Para as VLEs que não possuem ACGs deste tipo, é utilizada a ACG do tipo $[\psi_i: V_M \supset V_{LE}]$. Após a seleção, avaliando-se as restrições de interseção entre as VLEs e as ACGs selecionadas, as VLEs são alocadas nos conjuntos da partição.

Seja, novamente, o conjunto de ACGs de $Paciente_M$:

$ACGs(Paciente_M) = \{ \psi_1: Paciente_M \supset Paciente_1 ;$

$\psi_2: Paciente_M \supset Paciente_2 ;$

$\psi_3: Paciente_M[Paciente/Cidade='Fortaleza'] \equiv Paciente_1 \}$

O passo seguinte consiste da seleção das ACGs ψ_2 e ψ_3, pois ψ_2 é a única ACG de $Paciente_2$ e ψ_3 é a ACG de $Paciente_1$ que contempla o filtro da expressão de seleção de $Q_{Paciente}$ (Exp = Paciente/Cidade = 'Fortaleza'). Assim, as ACGs selecionadas são:

$\psi_2: Paciente_M \supset Paciente_2$ e

$\psi_3: Paciente_M[Paciente/Cidade='Fortaleza'] \equiv Paciente_1$

Seja a seguinte restrição de interseção entre as VLEs $Paciente_1$ e $Paciente_2$, inferida durante a geração da VM homogênea $Paciente_M$, conforme vimos no Capítulo V, e recuperada do catálogo do mediador:

$[r_1: Paciente_1 \cap Paciente_2 \neq \varnothing]$

Assim, de acordo com a definição de partição de integração, a partir das ACGs selecionadas, dos filtros da expressão de seleção Exp e da restrição de interseção, a partição $P_I(Q_{Paciente})$ é gerada conforme mostrado a seguir:

$$P_I(Q_{Paciente}) = \{ \underbrace{\{Paciente_1\}}_{P_{[\equiv]}}, \underbrace{\{Paciente_2\}}_{P_{[Exp]}}, \underbrace{\emptyset}_{P_{[\neg Exp]}} \}$$

Observe que a restrição de interseção r_1 informa que as VLEs $Paciente_1$ e $Paciente_2$ possuem interseção não vazia, ou seja, as informações de um mesmo paciente podem estar presentes nas duas bases. Caso $Paciente_1$ e $Paciente_2$ possuam ACGs de um mesmo tipo, elas devem ser alocadas em um mesmo subconjunto. Entretanto, como as ACGs de $Paciente_1$ e $Paciente_2$ são de tipos diferentes (≡ e ⊃), as VLEs foram alocadas em subconjuntos diferentes. Observe ainda que, como $Paciente_2$ possui a propriedade Paciente/Cidade, sobre a qual foi aplicado o filtro de Exp, a VLE $Paciente_2$ foi alocada no subconjunto $P_{[Exp]}$.

- **Geração do *Workflow* Inicial – $\mathcal{W}_i(Q_{Paciente})$**

O *workflow* inicial $\mathcal{W}_i(Q_{Paciente})$ é gerado a partir da partição $P_I(Q_{Paciente})$. Conforme apresentado na Seção 6.2.3, o *workflow* inicial só pode ser gerado se a condição essencial (Regra 6.1) for satisfeita. Para o exemplo, como o subconjunto $P_{[\equiv]}$ possui a VLE $Paciente_1$, a condição é satisfeita. Assim, o *workflow* inicial $\mathcal{W}_i(Q_{Paciente})$ assume a configuração do Caso 1 (ver Seção 6.2.3), conforme exibido na Figura 6.10.

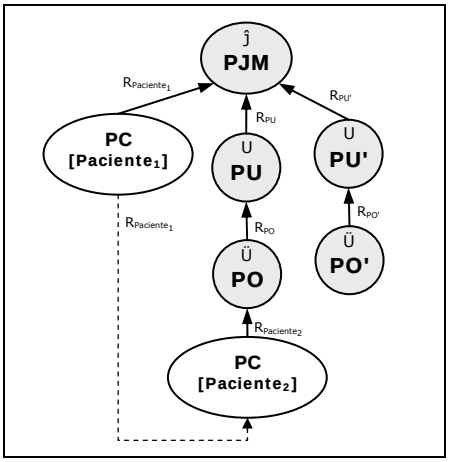


Figura 6.10 – Grafo do *workflow* inicial $\mathcal{W}_i[Q_{Paciente}]$

Observe na partição $P_I(Q_{Paciente})$ que $P_{[Exp]}$ contém apenas a VLE $Paciente_2$ e $P_{[\neg Exp]}$ é vazio. Desta forma, existem nós no grafo que representam operações de integração irrelevantes, como os processos PU, PO, PU' e PO' (vide Figura 6.10). Estes nós serão eliminados durante a geração do *workflow* final.

- **Geração do *Workflow* Final – $\mathcal{W}_f(Q_{Paciente})$**

O objetivo deste passo é adicionar ao *workflow* inicial $\mathcal{W}_i(Q_{Paciente})$ os processos responsáveis por consultar e integrar informações das VLEs Secundárias relevantes, resultando no *workflow* final $\mathcal{W}_f(Q_{Paciente})$. As subconsultas que deverão ser executadas sobre as VLEs Primárias e Secundárias são atribuídas aos Processos-Consulta também neste passo.

A seguir, é mostrada a seqüência de operações realizadas pelo algoritmo para a geração do *workflow* final $\mathcal{W}_f(Q_{Paciente})$ e a atribuição das subconsultas aos Processos-Consultas. O algoritmo realiza uma visita em profundidade (semelhante a uma busca em profundidade em um grafo) aos nós do grafo que representa o *workflow* $\mathcal{W}_i(Q_{Paciente})$, iniciando pelo nó-raiz. Para o exemplo dado, o nó-raiz é o nó PJM. Assim:

Operação 1: O nó PJM (nó-raiz) é visitado.

Operação 2: O nó PC[$Paciente_1$] é visitado como o primeiro nó-filho do nó PJM (vide Figura 6.11(a)). Como PC[$Paciente_1$] é um nó que representa um Processo-Consulta à VLE Primária $Paciente_1$, são realizadas operações para atribuir a sua subconsulta e incluir os processos de consulta às VLEs Secundárias ligadas à $Paciente_1$.

Operação 2.1: A subconsulta Q_1 é atribuída ao processo PC[$Paciente_1$], agora chamado PC[$Paciente_1$, Q_1].

Operação 2.2: É verificado no grafo das VLEs relevantes (vide Figura 6.8(b)) que a VLE $Paciente_1$ possui uma ligação para VLE Secundária $Patologia_3$. Então, o nó PC[$Paciente_1$, Q_1] é substituído no grafo pelo nó PJM_1 e são adicionados os filhos deste nó (vide Figura 6.11(b)), a saber: (i) o nó PC[$Paciente_1$, Q_1] e (ii) o nó PC[$Patologia_3$], a direita do nó PC[$Paciente_1$, Q_1]. O nó PC[$Patologia_3$] é recursivamente visitado.

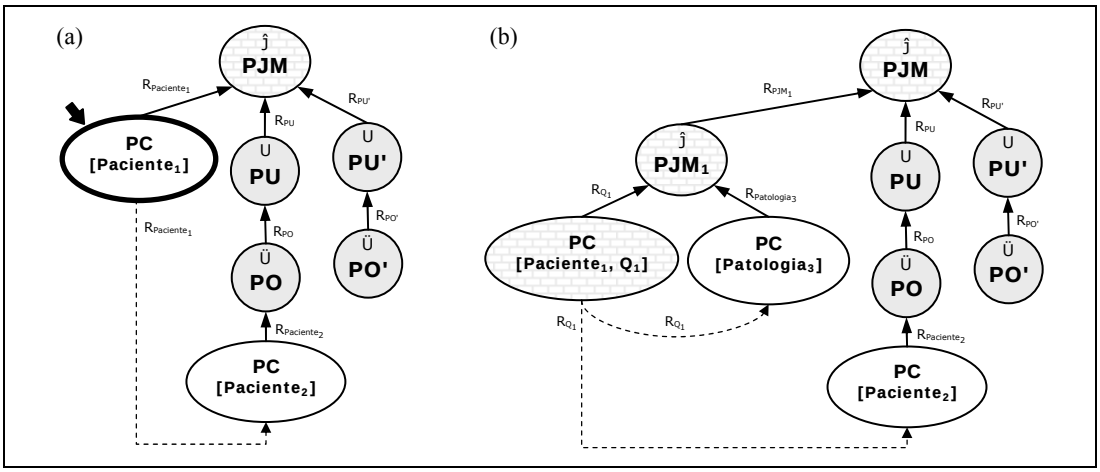


Figura 6.11 – (a) Nó visitado PC[$Paciente_1$]; (b) Inclusão do nó PC[$Patologia_3$]

Operação 2.3: O nó PC[Patologia$_3$] é visitado (vide Figura 6.12(a)). Como PC[Patologia$_3$] é um nó que representa um Processo-Consulta à VLE Secundária Patologia$_3$, são realizadas operações para atribuir a sua subconsulta e incluir os processos de consulta às VLEs Secundárias ligadas à Patologia$_3$. De acordo com o grafo de VLEs relevantes, a VLE Patologia$_3$ não possui ligação com VLEs Secundárias. Assim, apenas a subconsulta Q_3 é atribuída ao processo PC[Patologia$_3$], agora chamado PC[Patologia$_3$, Q_3].

Para seguir a estratégia baseada em semi-junção, o algoritmo adiciona à subconsulta Q_3 uma cláusula *where*, que filtra o resultado de acordo com um conjunto de valores (projeção do atributo de junção) que será extraído do resultado do nó PC[Paciente$_1$, Q_1]. A subconsulta Q_3 passa a ser um consulta parametrizada (vide Figura 6.12(b)) que busca de Patologia$_3$ apenas informações de patologias presentes no resultado de Q_1 (reduzindo o volume de dados trafegados).

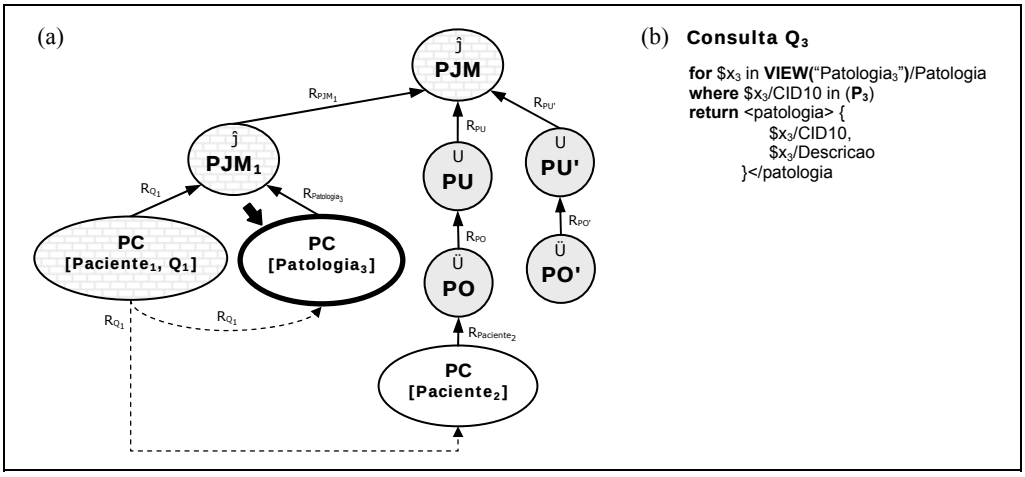


Figura 6.12 – (a) Nó visitado PC[Patologia$_3$]; (b) Consulta Q_3

Operação 3: O nó PU é visitado. Como este nó possui apenas um nó-filho para fornecer a entrada para a operação de união, ele é substituído pelo seu nó-filho PO.

Operação 4: O nó PO é visitado. Como este nó possui apenas um nó-filho para fornecer a entrada para a operação de união externa, ele é substituído pelo seu nó-filho PC[Paciente$_2$].

Operação 5: O nó PC[Paciente$_2$] é visitado (vide Figura 6.13(a)). Como PC[Paciente$_2$] é um nó que representa um Processo-Consulta à VLE Primária Paciente$_2$, são realizadas operações para atribuir a sua subconsulta e incluir os processos de consulta às VLEs Secundárias que ligadas à Paciente$_2$. De acordo com o grafo de VLEs relevantes, a VLE Paciente$_2$ não possui ligação com VLEs Secundárias. Assim, apenas a subconsulta Q_2 é atribuída ao processo PC[Paciente$_2$] , agora chamado PC[Paciente$_2$, Q_2].

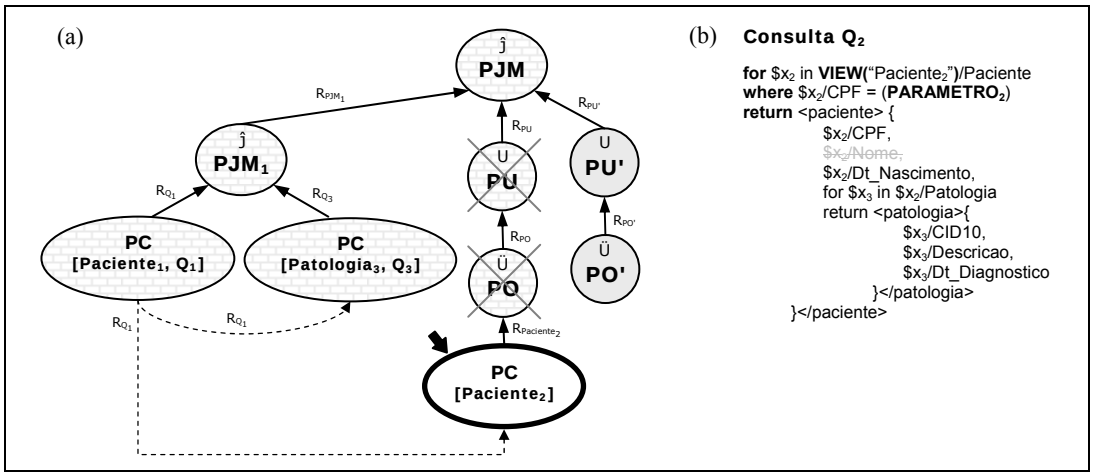


Figura 6.13 – (a) Nó visitado PC[Paciente$_2$]; (b) Consulta Q_2

As propriedades simples já selecionadas na consulta local do nó PC[Paciente$_1$, Q_1], como é o caso de Paciente/Nome, não serão novamente buscadas em Paciente$_2$ (exceção para as propriedades chaves-primárias, no caso, a propriedade Paciente/CPF). Isso acontece pois o algoritmo identifica que os resultados dos processos PC[Paciente$_1$, Q_1] e PC[Paciente$_2$, Q_2] serão integrados por uma operação de junção (PJM) e, portanto, dados redundantes não deverão ser consultados (redução do volume de dados trafegados).

Novamente, para seguir a estratégia baseada em semi-junção, o algoritmo adiciona à subconsulta Q_2 uma cláusula *where*, que filtra o resultado de acordo com um conjunto de valores que será extraído do resultado do nó PC[Paciente$_1$, Q_1]. A cláusula *where* anteriormente existente em Q_2 é removida. A subconsulta Q_2 passa a ser um consulta parametrizada (vide Figura 6.13(b)) que busca de Paciente$_2$ apenas informações de pacientes retornados no resultado de Q_1.

Operação 6: O nó PU' é visitado. Como este nó possui apenas um nó-filho para fornecer a entrada para a operação de união, ele substituído pelo seu nó-filho PO'.

Operação 7: O nó PO' é visitado. Como este nó não possui nós-filhos que forneçam as entradas para a operação de união externa, ele é excluído do grafo.

Ao final da visita aos nós do grafo do *workflow* $\mathcal{W}_i(Q_{Paciente})$, da inclusão dos nós de consulta e integração das informações das VLEs Secundárias relevantes e da atribuição das subconsultas, está gerado o *workflow* final de execução de $Q_{Paciente}$, chamado $\mathcal{W}_f(Q_{Paciente})$, conforme mostra a Figura 6.14.

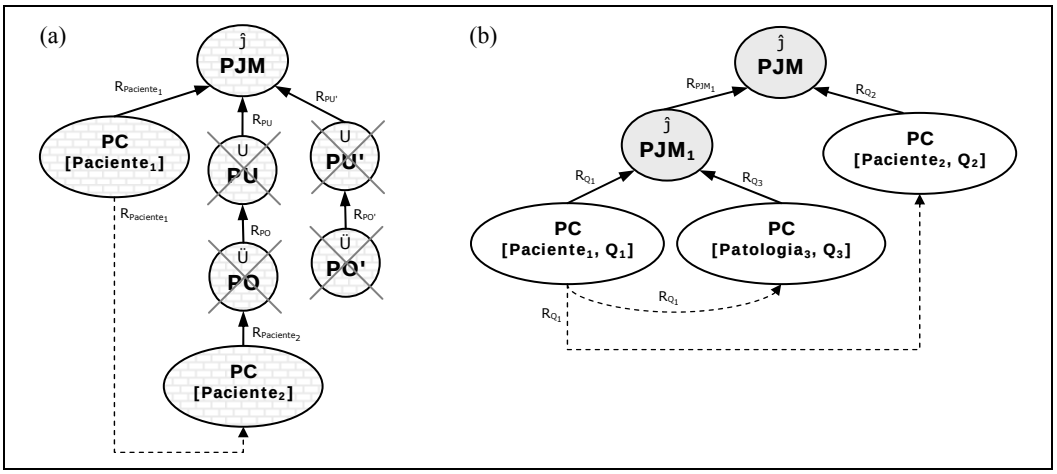


Figura 6.14 – (a) Grafo do *workflow* inicial $\mathcal{W}_i[Q_{Paciente}]$; (b) Grafo do *workflow* final $\mathcal{W}_f[Q_{Paciente}]$

Uma descrição informal da semântica do *workflow* de execução da consulta $Q_{Paciente}$ é a seguinte:

Inicialmente, os "pacientes de Fortaleza" são obtidos de $Paciente_1$ (consulta Q_1). Em seguida, observando-se que foi selecionada a propriedade Patologia/Descricao e que a propriedade não está disponível em $Paciente_1$, consulta-se $Patologia_3$ (consulta Q_3) para se obter aquela propriedade, já que $Patologia_3$ está ligada a $Paciente_1$, conforme é verificado no grafo das VLEs relevantes. É realizada uma operação de junção ($\hat{J}$) para integrar os resultados de Q_1 e Q_3. Por último, $Paciente_2$ é consultada (consulta Q_2) para buscar informações complementares dos pacientes retornados por Q_1 que não são encontradas em $Paciente_1$, como é o caso de Dt_Nascimento, ou mais alguma Patologia, que é uma propriedade multivalorada de $Paciente_M$. É realizada uma operação junção ($\hat{J}$) para integrar o resultado de Q_2 com o resultado da junção de Q_1 e Q_3.

Capítulo VII

Conclusões e Trabalhos Futuros

Neste trabalho, propomos um *framework* para sistemas de mediação baseados em XML. No *framework* proposto, o esquema global (ou esquema de mediação) consiste de um conjunto de visões XML que integram dados de fontes heterogêneas e distribuídas, chamadas de Visões de Mediação (VMs).

O *framework* adota uma arquitetura de três níveis de esquemas, onde uma VM (nível de mediação) é definida em termos de visões XML (nível exportado) que são exportadas pelas fontes de dados (nível local). Cada visão exportada é definida em termos de uma única fonte de dados e consiste de um fragmento (vertical, horizontal ou híbrido) da VM. Esta arquitetura objetiva, portanto, simplificar e melhorar o desempenho do processamento de consultas sobre as VMs, trazendo-o para um contexto de integração de dados de fontes homogêneas, uma vez que as visões XML exportadas pelas fontes são fragmentos homogêneos da VM.

As principais contribuições desta dissertação são:

1) Definição do *framework* para sistemas de mediação baseados em XML que adota uma arquitetura de três níveis de esquemas. Esta arquitetura é também definida neste trabalho.
2) Formalismo e processo para a especificação de VMs diretamente em termos de fontes de dados heterogêneas e distribuídas (VMs heterogêneas), incluindo um formalismo simples, mas expressivo, para a definição dos mapeamentos (Assertivas de Correspondência).
3) Algoritmo para, a partir da especificação de uma VM heterogênea, gerar automaticamente uma VM equivalente (VM homogênea) definida em termos de visões XML exportadas pelas fontes. As visões exportadas são também geradas por este algoritmo.
4) Metodologia (e o respectivo algoritmo) para o processamento de consultas sobre as VMs homogêneas. Esta metodologia é também adotada pelo *framework*.

É importante destacar que a metodologia proposta é diretamente favorecida pela arquitetura de três níveis de esquemas, ou seja, o fato das visões exportadas pelas fontes representarem fragmentos homogêneos da VM permite ao algoritmo que implementa a metodologia: (i) realizar uma decomposição simples de consultas, sem a necessidade de verificar mapeamentos de esquemas em tempo de execução; (ii) implementar a integração de resultados de subconsultas apenas com operações de união e junção; e (iii) adotar estratégias de otimização, como a geração de planos de execução de consultas com base na relevância e distribuição dos dados nas fontes, na redução do volume de dados trafegados e no custo computacional das operações de integração.

Como sugestões para trabalhos futuros, destacam-se:

(i) Implementar um ambiente para apoiar as atividades de especificação de VMs heterogêneas e a geração de VMs homogêneas. Esse ambiente seria composto pelas seguintes ferramentas[3]: (i) Editor de Esquema de uma VM; (ii) Editor de Assertivas; (iii) Editor de Restrições e Ligações entre as fontes de dados; e (iv) Gerador de VMs Homogêneas e VLEs, o qual implementaria o algoritmo **GeraVMHomogenea** proposto neste trabalho;

(ii) Implementar um sistema de mediação baseado no *framework* proposto, integrando os seus vários módulos. O mediador do sistema implementaria o algoritmo **ProcessaConsulta** proposto neste trabalho;

(iii) Estender o formalismo das assertivas de correspondência para possibilitar a especificação de novos tipos de visões XML de integração de dados;

(iv) Estudar a possibilidade e os impactos para tratar: (i) a qualidade dos dados das fontes e implementar uma resolução de conflitos mais avançada; (ii) a utilização de mecanismos para obter estatísticas e custos para apoiar a etapa de otimização global da metodologia proposta; (iii) a aplicação de filtros mais complexos (e funções XQuery) sobre as propriedades das VMs; (iv) a utilização de ontologias para a definição do esquema global, possibilitando inferir novos conhecimentos de integração e aumentar o desempenho do processamento de consultas.

[3] Algumas ferramentas, como o Editor de Esquema de uma VM e o Editor de Assertivas, já possuem versões anteriores [40][45] que poderão ser apenas evoluídas ou adaptadas neste trabalho.

Apêndice I

Algoritmos

Neste apêndice são apresentados os algoritmos para o processamento de consultas sobre VMs, de acordo a metodologia proposta neste trabalho. O algoritmo principal é o **ProcessaConsulta**, o qual invoca outros algoritmos com finalidades específicas na execução do processamento de uma consulta.

A1.1. Introdução

Considere o seguinte *workflow* de execução de uma consulta (plano de execução), representado pelo grafo exibido na Figura A1.15. As seguintes nomenclaturas são utilizadas pelos algoritmos, conforme é exemplificado a seguir:

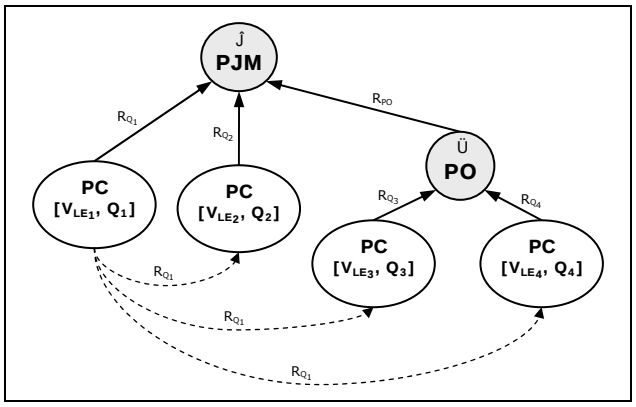


Figura A1.15 – Grafo exemplo

- *Nó-Raiz*: PJM é o nó-raiz do grafo.
- *Nó-Pai*: PJM é o nó-pai dos nós PC[V_{LE1}, Q_1], PC[V_{LE2}, Q_2] e PO.
- *Nó-Filho ou Nó-Fornecedor*: PC[V_{LE1}, Q_1], PC[V_{LE2}, Q_2] e PO são nós-filhos ou nós-fornecedores do nó PJM.
- *1º Nó-Fornecedor*: PC[V_{LE1}, Q_1] é o primeiro nó-fornecedor do nó PJM.
- *Nó-Integração*: PJM e PO são nós-integração.
- *Nó-Antecedente*: PJM e PO são nós-antecedentes dos nós PC[V_{LE3}, Q_3] e PC[V_{LE4}, Q_4]. O nó-raiz é nó-antecedente de qualquer outro nó do grafo.

A1.2. Algoritmos

Algoritmo: Processa_Consulta

Entrada: Q_M (Q_M é a consulta realizada sobre uma visão de mediação V_M).

Saída: C (coleção XML com o resultado de Q_M).

início

// Alguns algoritmos, por terem sido descritos no decorrer deste trabalho ou por serem triviais, não serão apresentados.

// 1. Localização dos Dados

// 1.1. Identificação das VLEs primárias relevantes

$G_{Relevantes}'$ ← **Identifica_VLEs_Primarias_Relevantes(**Q_M**);**

// 1.2. Identificação das VLEs secundárias relevantes

$G_{Relevantes}$ ← **Identifica_VLEs_Secundarias_Relevantes(**$G_{Relevantes}'$ **,** Q_M**);**

// 2. Decomposição de Q_M

Decompoe_Consulta($G_{Relevantes}$**);**

// 3. Otimização Global

// 3.1. Geração da partição de integração de Q_M ($P_I(Q_M)$)

P_I ← **Gera_Particao_Integracao_Consulta(**$G_{Relevantes}$ **,** Q_M**);**

// 3.2. Geração do workflow de execução inicial de Q_M ($\mathcal{W}_I(Q_M)$)

G_{QM}' ← **Gera_Workflow_Execucao_Inicial(**P_I**);**

// 3.3. Geração do workflow de execução final de Q_M ($\mathcal{W}_f(Q_M)$)

G_{QM} ← **Gera_Workflow_Execucao_Final(**G_{QM}' **,** Q_M **,** $G_{Relevantes}$**);**

// 4. Execução do plano de Q_M

C ← **Executa_Workflow(**G_{QM}**);** //conforme apresentado na seção 4.4.2

retorna C;

fim.

Algoritmo: Identifica_VLEs_Primarias_Relevantes

Entrada: Q_M (Q_M é a consulta realizada sobre uma visão de mediação V_M).

Saída: $G_{Relevantes}'$ ($G_{Relevantes}'$ é o grafo parcial das VLEs relevantes para Q_M, obtido a partir do grafo da hierarquia de VLEs de V_M, após a remoção dos nós (e seus nós-filhos) que representam VLEs Primárias não relevantes).

início

// Obtendo do catálogo do mediador o grafo da hierarquia de VLEs de V_M, o qual foi gerado na criação de V_M.

$G_{Relevantes}'$ ← **Obtem_Grafo_Hierarquia_VLEs(**Q_M**.ObtemVmConsultada());**

no_raiz ← $G_{Relevantes}'$**.Obtem_No_Raiz();**

// Obtendo os nós que representam VLEs Primárias, ou seja, os nós-filhos do nó-raiz do grafo da hierarquia das VLEs.

para i **de** 1 **até** no_raiz**.Obtem_Qtd_Nos_Filhos() faça**

no_vle_primaria ← no_raiz**.Obtem_No_Filho(**i**);** // i é o índice do nó-filho

// 1º critério: verifica se há assertiva de correspondência de global ou de coleção (ACG) que determina que a VLE Primária
// não possui informações relevantes, considerando os filtros de seleção aplicados em Q_M (expressão Exp), ou seja,
// verifica se existe assertiva do tipo [V_M[Exp] ∩ V_{LE} = ∅] ou [V_M[Exp'] ∩ V_{LE} = ∅], onde Exp' é parte de Exp.
// Caso positivo, marca o nó para remoção do grafo parcial das VLEs relevantes (e por conseqüência os seus nós-filhos).

se Existe_ACE_VLE_Nao_Relevante(no_vle_primaria**.Obtem_VLE(),** Q_M**) então**

no_raiz**.Marca_No_Remocao(**no_vle_primaria**);**

senão

// 2º critério: verifica se o esquema consolidado da VLE Primária possui informações relevantes para a consulta,
// ou seja, se possui alguma propriedade selecionada por Q_M ou alguma propriedade onde foi aplicado filtro de seleção
// em Q_M. Caso negativo, remove o nó (e por conseqüência os seus nós-filhos) que representa a VLE Primária.

```
        // Para a comparação de tipos, adiciona ao tipo de Q_M as propriedades onde foram aplicados filtros de seleção.
        // Assim, estas propriedades são também tratadas como relevantes para a consulta.
        Q' ← Q_M;
        Q'.Adiciona_Propriedades_Expressao_Selecao();
        // Obtém do catálogo do mediador o esquema consolidado da VLE Primária.
        tipo_consolidado_vle ← Obtem_Tipo_Consolidado(no_vle_primaria.Obtem_VLE());
        Compara_Tipos(Q'.Obtem_Tipo(), tipo_consolidado_vle);
        // Verifica se existe alguma propriedade selecionada em Q' que é suportada pelo esquema consolidado.
        se não(Q'.Existe_Propriedade_Suportada()) então
          no_raiz.Marca_No_Remocao(no_vle_primaria);
        fim se;
      fim se;
    fim para;
    G_Relevantes'.Remove_Nos_Marcados_Remocao();
    retorna G_Relevantes';
fim.
```

Algoritmo: Identifica_VLEs_Secundarias_Relevantes

Entrada: $G_{Relevantes}'$, Q_M ($G_{Relevantes}'$ é o grafo parcial das VLEs relevantes para Q_M , obtido no algoritmo Identifica_VLEs_Primarias_Relevantes. Q_M é a consulta realizada sobre uma visão de mediação V_M).

Saída: $G_{Relevantes}$ ($G_{Relevantes}$ é o grafo das VLEs relevantes para Q_M, obtido a partir do grafo parcial, após a remoção dos nós que representam VLEs Secundárias que não são relevantes para Q_M).

```
início
    // Visita os nós que permanecem no grafo parcial das VLEs relevantes para identificar e remover os nós que
    // representam VLEs Secundárias que não são relaventes para Q_M. As visitas são iniciadas pelos nós que representam
    // VLEs Primárias, ou seja, os nós-filhos do nó-raiz do grafo parcial das VLEs relevantes.
    no_raiz ← G_Relevantes'.Obtem_No_Raiz();
    para i de 1 até no_raiz.Obtem_Qtd_Nos_Filhos() faça
      no_vle_primaria ← no_raiz.Obtem_No_Filho(i); // i é o índice do nó-filho
      // Visita cada nó que representa uma VLE Primária.
      // Se for o caso:
      //  i. Assinala o nó em questão com marcação de filtro, pois o nó (ou seus nós-descendentes) possui propriedades
      //  onde foram aplicados filtros de seleção em Q_M;
      // ii. Visita (chamada recursiva) os seus nós-filhos para obter propriedades não suportadas pela VLE do nó em questão.
      // Em cada visita, é gerado o objeto consulta local, o qual será posteriormente utilizado para gerar a subconsulta XQuery
      // sobre a VLE do nó em questão.
      Visita_No_Grafo_VLEs_Relevantes(no_vle_primaria, Q_M);
    fim para;
    // Remove os nós não visitados, pois representam VLEs Secundárias que não são relevantes para a consulta.
    G_Relevantes ← G_Relevantes';
    G_Relevantes.Remove_Nos_Nao_Visitados();
    retorna G_Relevantes;
fim.
```

Algoritmo: Visita_No_Grafo_VLEs_Relevantes

Entrada: no_atual, Q_M (no_atual é um nó do grafo parcial de VLEs relevantes que é visitado e gerado o objeto consulta local. Se for o caso, o nó é assinalado com uma marcação de filtro, pois representa uma VLE que possui propriedades onde foram aplicados filtros de seleção em Q_M. Neste caso, os nós-antecedentes de no_atual também são assinalados com marcações de filtro).

Saída: Ø (gerado o objeto consulta local e, se for o caso, assinala o `no_atual` e os seus nós-antecedentes com uma marcação de filtro).

início

```
// Nota: Além de assinalar as marcações de filtro, este algoritmo é chamado recursivamente, em alguns casos,
//  para visitar os nós-filhos do no_atual e identificar propriedades selecionadas em Q_M que não são suportadas pela
// VLE do no_atual, mas que são suportadas pelas VLEs destes nós-filhos. Assim, estes nós, os quais representam
// VLEs Secundárias, são também visitados e identificados como relevantes para a consulta.

// Marca no-atual como visitado.
no_atual.Marca_Visitado();

// Para a comparação de tipos, adiciona ao tipo de Q_M as propriedades onde foram aplicados filtros de seleção.
// Assim, estas propriedades são também tratadas como relevantes para a consulta.
Q' ← Q_M;
Q'.Adiciona_Propriedades_Expressao_Selecao();
Compara_Tipos(Q'.Obtem_Tipo(), no_atual.Obtem_VLE().Obtem_Tipo());

// A partir das propriedades marcadas como suportadas em Q' (incluindo as propriedades suportadas onde foram
// aplicados filtro de seleção), gera o objeto consulta local e o atribui ao no_atual.
no_atual.Atribui_Consulta_Local(Q'.Gera_Consulta_Local());

// Verifica se alguma propriedade onde foi aplicado filtro de seleção é suportada pela VLE do no_atual.
// Caso positivo, assinala o no_atual e todos os seus nós-antecedentes com marcações de filtro.
se Q'.Existe_Propriedade_Expressao_Selecao_Suportada() então
  no_atual.Assinala_Marcacao_Filtro(); // este método assinala também os nós-antecedentes
fim se;

// Verifica se alguma propriedade complexa de Q' não é suportada totalmente (todas as suas subpropriedades)
// pela VLE do no_atual. Caso positivo, é iniciada uma busca por informações complementares nos nós-filhos que
// representam VLEs Secundárias. As propriedades simples de Q' que não são suportadas pela VLE do no_atual
// são descartadas.
se Q'.Existe_Propr_Complexa_Nao_Suportada_Total() então
  para cada p ∈ Q'.Obtem_Propr_Complexas_Nao_Suportadas_Total () faça
    // Verifica se o no_atual tem algum nó-filho cuja aresta possui uma ligação (ℓ) que envolve a propriedade complexa
    // não suportada totalmente.
    se no_atual.Existe_No_Filho_Ligacao_Propriedade(p) então
      // Q'' é uma nova consulta representada pelo tipo da propriedade complexa não suportada totalmente,
      // retirando-se, com exceção das chaves, as subpropriedades já encontradas na VLE do nó atual e
      // que foram marcadas como suportadas no tipo daquela propriedade.
      Q'' ← p.Obtem_Tipo_Com_Marcacoes_Suportadas();
      Q''.Remove_Propriedades_Suportadas();
      no_vle_secundaria ← no_atual.Obtem_No_Filho_Ligacao_Propriedade(p);
      Visita_No_Grafo_VLEs_Relevantes(no_vle_secundaria, Q''); // Chamada recursiva
    fim se;
  fim para;
fim se;
fim.
```

Algoritmo: Compara_Tipos

Entrada: T_{XML_1}, T_{XML_2} (T_{XML_1} e T_{XML_2} são os tipos XML que serão comparados. Cada propriedade de T_{XML_1} que está presente em T_{XML_2} é marcada em T_{XML_1} como suportada. Para propriedades complexas, o algoritmo é chamado recursivamente).

Saída: Ø (marca em T_{XML_1} as propriedades que são suportadas por T_{XML_2}).

início

```
// A comparação dos tipos é realizada pela verificação de quais propriedades de T_XML_1 estão presentes em T_XML_2. Como os
// tipos possuem estruturas compatíveis e não há duplicidade de nomes de propriedades, a verificação de pertinência é feita
```

```
  // pelos nomes das propriedades. No caso de propriedades complexas, seus tipos são também comparados, em uma chamada
  // recursiva ao algoritmo Compara_Tipos.
  para cada p ∈ T_XML_1.Obtem_Propriedades() faça
    se p.ehTipoSimples() então
      se p.nome() ∈ T_XML_2.Obtem_Nomes_Propriedades_Simples() então
        p.Marca_Como_Suportada();
      fim se;
    senão
      se p.nome() ∈ T_XML_2.Obtem_Nomes_Propriedades_Complexas() então
        p' ← T_XML_2.Obtem_Propriedade(p.nome());
        Compara_Tipos(p.Obtem_Tipo(), p'.Obtem_Tipo()); // Chamada recursiva
      fim se;
    fim se;
  fim para;
fim.
```

Algoritmo: Decompoe_Consulta

Entrada: $G_{Relevantes}$ ($G_{Relevantes}$ é o grafo das VLEs relevantes para Q_M).

Saída: Ø (gera e atribui aos nós de $G_{Relevantes}$ as subconsultas XQuery que deverão ser executadas sobre as VLEs e gera a expressão global de execução de Q_M).

```
início
  // Visita os nós do grafo das VLEs relevantes. As visitas são iniciadas pelos nós que representam
  // VLEs Primárias, ou seja, os nós-filhos do nó-raiz do grafo das VLEs relevantes.
  no_raiz ← G_Relevantes.Obtem_No_Raiz();
  para i de 1 até no_raiz.Obtem_Qtd_Nos_Filhos() faça
    // Visita cada nó que representa uma VLE Primária e, a partir do objeto consulta local gerado na etapa de localização
    // dos dados, gera a subconsulta XQuery correspondente.
    no_vle_primaria ← no_raiz.Obtem_No_Filho(i); // i é o índice do nó-filho
    Gera_Subconsulta_Local_XQuery(no_vle_primaria);
  fim para;
  // Gera a expressão global de execução de Q_M, a partir do grafo das VLEs relevantes e das subconsultas que foram
  // geradas e armazenadas neste grafo.
  Gera_Expressao_Global_Execucao(G_Relevantes);
fim.
```

Algoritmo: Gera_Subconsulta_Local_XQuery

Entrada: no_atual (no_atual é um nó do grafo das VLEs relevantes que é visitado para a geração e atribuição da subconsulta XQuery que deverá ser executada sobre a VLE correspondente).

Saída: Ø (gera e atribui ao no_atual e aos nós descendentes (por chamadas recursivas) suas subconsultas em XQuery).

```
início
  // Obtendo o objeto consulta local gerado na etapa de localização dos dados.
  Q_L ← no_atual.Obtem_Consulta_Local();
  no_atual.Atribui_Subconsulta_XQuery(Q_L.Gera_Consulta_XQuery());
  // Visita os nós descendentes do no_atual para gerar e atribuir as subconsulta XQuery correspondentes.
  para i de 1 até no_atual.Obtem_Qtd_Nos_Filhos() faça
    no_vle_secundaria ← no_raiz.Obtem_No_Filho(i); // i é o índice do nó-filho
    Gera_Subconsulta_Local_XQuery(no_vle_secundaria); // Chamada recursiva
  fim para;
fim.
```

Algoritmo: Gera_Particao_Integracao_Consulta

Entrada: $G_{Relevantes}$, Q_M ($G_{Relevantes}$ é o grafo das VLEs relevantes para a consulta Q_M).

Saída: P_I (P_I é a partição de integração do conjunto das VLEs Primárias relevantes para Q_M).

```
início
  // Obtém do grafo das VLEs relevantes o conjuntos de VLEs Primárias relevantes
  vles_primarias ← G_Relevantes.Obtem_VLEs_Primarias();
  // Iniciando os subconjuntos da partição de integração
  P[≡] ← ∅;
  P[Exp] ← ∅;
  P[¬Exp] ← ∅;
  // Alocando as VLEs Primárias relevantes nos subconjuntos da partição de integração, de acordo com:
  //  i. As ACGs entre a VM e as VLEs Primárias;
  //  ii. Os filtros de seleção (expressão Exp) aplicados em Q_M ;
  // iii. A existência de interseção entre as VLEs Primárias;
  para cada vle ∈ vles_primarias faça
    // 1. VLEs do subconjunto P[≡]
    // Verifica se existe assertiva de correspondência de global ou de coleção (ACG) do tipo [ V_M[Exp] ≡ vle ]; ou
    // se existe ACG do tipo [ V_M ≡ vle ] e o esquema consolidado de vle suporta todas as propriedades
    // onde foram aplicados filtros de seleção em Q_M (suporta totalmente a expressão Exp).
    // Obtém do catálogo do mediador o esquema consolidado da VLE Primária.
    tipo_consolidado_vle ← Obtem_Tipo_Consolidado(vle);
    se ( Existe_ACE_Equivalencia_Expr_Selecao(vle, Q_M) ou
         ( Existe_ACE_Equivalencia(vle, Q_M) e
           Tipo_Suporta_Total_Expr_Selecao(tipo_consolidado_vle, Q_M) ) ) então
      P[≡].Adiciona_VLE(vle);
    // 2. VLEs do subconjunto P[Exp]
    // Verifica se o esquema consolidado de vle suporta pelo menos uma das propriedades onde foram aplicados
    // filtros de seleção em Q_M (suporta parcialmente a expressão Exp).
    senão se Tipo_Suporta_Parcial_Expr_Selecao(tipo_consolidado_vle, Q_M) então
      // Adiciona vle em subconjuntos P[Exp]_i de P[Exp], de acordo com a existência de interseção entre vle e
      // as VLEs que já foram adicionadas. A interseção é verificada através das restrições de interseção recuperadas
      // do catálogo do mediador.
      P[Exp].Adiciona_VLE_Considerando_Intersecao(vle);
    // 3. VLEs do subconjunto P[¬Exp]
    senão
      // Semelhante ao conjunto P[Exp], adiciona vle em subconjuntos P[¬Exp]_j de P[¬Exp], de acordo com a existência
      // de interseção entre vle e as VLEs que já foram adicionadas.
      P[¬Exp].Adiciona_VLE_Considerando_Intersecao(vle);
    fim se;
  fim para;
  P_I ← { P[≡] , P[Exp] , P[¬Exp] };
  retorna P_I;
fim.
```

Algoritmo: Gera_Workflow_Execucao_Inicial

Entrada: P_I (P_I é a partição de integração do conjunto das VLEs Primárias relevantes para uma consulta Q_M).

Saída: G_{QM}' (G_{QM} é o grafo que representa o workflow de execução inicial de uma consulta Q_M).

```
início
  // Obtendo os subconjuntos da partição de integração P_I
  P[≡] ← P_I.Obtem_Subconjunto('≡');
  P[Exp] ← P_I.Obtem_Subconjunto('Exp');
  P[¬Exp] ← P_I.Obtem_Subconjunto('¬Exp');

  // Aplicando a Regra 6.1 – Condição essencial para a geração do workflow de execução inicial
  se P[≡] = ∅ então
    // Verificando se algum subconjunto P[Exp]_i de P[Exp] suporta totalmente a expressão de seleção de Q_M
    suporta_total ← falso;
    para cada P[Exp]_i ∈ P[Exp] faça
      se P[Exp]_i.Suporta_Total_Expr_Selecao(Q_M) então
        suporta_total ← verdadeiro;
        volta_inicio_do_laço; // Volta para o início do laço, com o próximo valor de P[Exp]i
      senão
        // Aplicando a Regra 6.2 – Exclusão de subconjuntos P[Exp]_i irrelevantes para Q_M em P_I(Q_M).
        P[Exp].Exclui_Subconjunto(P[Exp]_i);
      fim se;
    fim para;
    // Caso não exista subconjunto P[Exp]_i que suporta totalmente Exp, o workflow não poderá ser gerado.
    se não suporta_total então
        retorna ∅;
    fim se;
  fim se;

  // Gerando o nó-raiz do workflow inicial
  no_juncao_multipla ← Gera_No_Junção_Múltipla();

  // Gerando o subgrafo do subconjunto P[≡]
  se P[≡] ≠ ∅ então
    primeira_vle ← verdadeiro;
    para cada vle ∈ P[≡] faça
      no_consulta ← Gera_No_Consulta(vle);
      no_juncao_multipla.Adiciona_No_Filho(no_consulta);
      // Selecionando o nó que fornecerá os valores dos parâmetros dos demais nós-consulta. Este nó filtra apenas as
      // entidades que satisfazem a expressão Exp.
      se primeira_vle então
        no_fornecedor_param ← no_consulta;
        primeira_vle ← falso;
      senão
        no_consulta.Adiciona_No_Fornece_Param(no_fornecedor_param);
      fim_se;
    fim_para;
  fim_se;

  // Gerando o subgrafo do subconjunto P[Exp]
  no_union_exp ← Gera_No_Union();
```

```
se P[Exp] ≠ ∅ então
  para cada P[Exp]i ∈ P[Exp] faça
    no_outer_union ← Gera_No_Outer_Union();
    // Gerando o nó-filtro, com os filtros de Exp, para filtrar apenas as entidades que satisfazem a expressão Exp.
    // O nó-filtro é gerado apenas quando o subconjunto P[≡] é vazio, pois, neste caso, ainda não foram obtidas das
    // VLEs deste subconjunto as entidades desejadas, ou seja, as entidades que satisfazem Exp.
    se P[≡] = ∅ então
      no_filtro ← Gera_No_Filtro(QM);
      no_filtro.Adiciona_No_Filho(no_outer_union);
      no_union_exp.Adiciona_No_Filho(no_filtro);
    senão
      no_union_exp.Adiciona_No_Filho(no_outer_union);
    fim_se;
    para cada vle ∈ P[Exp]i faça
      no_consulta ← Gera_No_Consulta(vle);
      no_consulta.Adiciona_No_Fornece_Param(no_fornecedor_param);
      no_outer_union.Adiciona_No_Filho(no_consulta);
    fim_para;
  fim_para;
  // Selecionando o nó que fornecerá os valores dos parâmetros dos demais nós-consulta. Este nó filtra apenas as
  // entidades que satisfazem a expressão Exp.
  se no_fornecedor_param = nulo então
    no_fornecedor_param ← no_union_exp;
  fim_se;
fim_se;

// Gerando o subgrafo do subconjunto P[¬Exp]
no_union_nao_exp ← Gera_No_Union();
se P[¬Exp] ≠ ∅ então
  para cada P[¬Exp]j ∈ P[¬Exp] faça
    no_outer_union ← Gera_No_Outer_Union();
    no_union_nao_exp.Adiciona_No_Filho(no_outer_union);
    para cada vle ∈ P[¬Exp]j faça
      no_consulta ← Gera_No_Consulta(vle);
      no_consulta.Adiciona_No_Fornece_Param(no_fornecedor_param);
      no_outer_union.Adiciona_No_Filho(no_consulta);
    fim_para;
  fim_para;
fim_se;
no_juncao_multipla.Adiciona_No_Filho(no_union_exp);
no_juncao_multipla.Adiciona_No_Filho(no_union_nao_exp);
GQM' ← Gera_Grafo(no_juncao_multipla);
retorna GQM';
fim.
```

Algoritmo: Gera_Workflow_Execucao_Final

Entrada: G_{QM}', Q_M , $G_{Relevantes}$ (G_{QM}' é o grafo que representa o workflow de execução inicial de Q_M e $G_{Relevantes}$ é o grafo de VLEs relevantes para Q_M).

Saída: G_{QM} (G_{QM} é o grafo que representa o workflow de execução final da consulta Q_M).

início

```
// Remove todos os nós que deveriam realizar operações de integração mas que não possuem nós-fornecedores
// ou possuem apenas um único nó-fornecedor. Neste último caso, o único nó-fornecedor assume o lugar do seu nó-pai.
G_QM ← G_QM';
G_QM.Remove_Nos_Integracao_Desnecessarios();

// Selecionando as chaves das propriedades complexas que foram parcialmente selecionadas em Q_M.
// As chaves das propriedades complexas são necessárias para a realização das operações de integração.
// Ao final do processamento, um filtro é aplicado para remover as chaves que não foram originalmente selecionadas em Q_M.
Q ← Q_M;
Q.Seleciona_Chaves_Propriedades_Complexas();

// Visita os nós do grafo do workflow de execução inicial e, para cada nó que representa um Processo-Consulta a uma
// VLE Primária, verifica no grafo das VLEs relevantes se existem VLEs Secundárias hierarquicamente ligadas à VLE Primária.
// Caso positivo, adiciona os nós que representam os processos responsáveis por consultar e integrar as informações das
// VLEs Secundárias. No momento da visita a um nó que representa um Processo-Consulta a uma VLE Primária ou da inclusão
// de nó que representa um Processo-Consulta a uma VLE Secundária, são atribuídas (ou novamente geradas e atribuídas)
// a estes processos as respectivas subconsultas XQuery (decomposição de Q_M em subconsultas sobre as VLEs relevantes)
// que deverão ser enviadas aos serviços web locais para execução sobre as VLEs.
no_raiz ← G_QM.Obtem_No_Raiz();
Visita_No(no_raiz, Q_M, G_Relevantes);

retorna G_QM;
```

fim.

Algoritmo: Visita_No

Entrada: no_visitado, Q, $G_{Relevantes}$ (no_visitado é o nó que será visitado, Q é uma consulta sobre uma visão XML (pode ser sobre a VM ou VLEs) e $G_{Relevantes}$ é o grafo das VLEs relevantes).

Saída: Ø

início

```
// De acordo com o tipo de nó visitado é chamado o algoritmo de visita adequado.
se no_visitado.ehOperador() então
    // Processo Junção Múltipla (Ĵ)
    se no_visitado.valor() = "PJM" então
        Visita_No_Juncao_Multipla(no_visitado, Q, G_Relevantes);
    // Processo Union (U) ou Processo Outer Union (Ü)
    senão se no_visitado.valor() = "PU" ou no_visitado.valor() = "PO" então
        Visita_No_Union_Outer_Union(no_visitado, Q, G_Relevantes);
    // Processo Filtro (F), a visita é repassada ao seu primeiro e único nó-fornecedor, que é um Processo Outer Union.
    senão se no_visitado.valor() = "PF" então
        Visita_No_Union_Outer_Union(no_visitado.Obtem_No_Fornecedor(1), Q, G_Relevantes);
    fim se;
senão
    // O nó representa um Processo-Consulta a uma VLE Primária. O nó será visitado e será atribuída (ou novamente gerada e
    // atribuída) a subconsulta XQuery sobre a VLE Primária. A subconsulta será armazenada como atributo do Processo-Consulta.
    // De acordo com o grafo das VLEs relevantes, serão acrescentados os processos responsáveis por consultar e integrar as
    // informações das VLEs Secundárias ligadas à VLE Primária,.
    Visita_No_Consulta(no_visitado, Q, G_Relevantes);
fim se; fim.
```

Algoritmo: Visita_No_Juncao_Multipla

Entrada: no_pjm, Q, $G_{Relevantes}$ (no_pjm é o nó que será visitado, Q é uma consulta sobre uma visão XML (pode ser sobre a VM ou VLEs) e $G_{Relevantes}$ é o grafo das VLEs relevantes).

Saída: Ø

início

```
// Quando o nó visitado é um nó-junção-múltipla, à medida que as propriedades selecionadas em Q vão sendo encontradas
// em nós-fornecedores do tipo nó-consulta, estas propriedades são eliminadas (exceto as chaves) do tipo de Q . Assim, na
// visita ao próximo nó-fornecedor do nó-junção-múltipla, as propriedades já encontradas não são novamente buscadas, evitando
// o tráfego desnecessário de informações. Para estes casos, as subconsultas dos nós-consulta posteriormente visitados serão
// novamente geradas, desconsiderando-se as subconsultas correspondentes geradas na fase de decomposição de Q_M.
Q' ← Q;

// Visita os nós-fornecedores do nó-junção-múltipla
para i de 1 até no_pjm.Obtem_Qtd_Nos_Fornecedores() faça
  no_fornecedor ← no_pjm.Obtem_No_Fornecedor(i);
  se Q'.Existe_Propriedade_Nao_Suportada() então // chaves das propriedades complexas não são
    Visita_No(no_fornecedor, Q', G_Relevantes);      // consideradas neste método.
    // Elimina da consulta propriedades que já foram encontradas em outras VLEs, exceto chaves marcadas, caso o
    // nó-fornecedor seja do tipo nó-consulta.
    se não no_fornecedor.ehOperador() então
      Q'.Remove_Propriedades_Suportadas();
    fim se;
  senão
    // Se todas as propriedades já foram encontradas, os demais nós-fornecedores do nó-junção-múltipla também serão
    // descartados, evitando o tráfego desnecessário de informações.
    no_pjm.Remove_No(no_fornecedor);
  fim se;
fim para;
fim.
```

Algoritmo: Visita_No_Union_Outer_Union

Entrada: no_union_outer, Q, $G_{Relevantes}$ (no_union_outer é o nó que será visitado, Q é uma consulta sobre uma visão XML (pode ser sobre a VM ou VLEs) e $G_{Relevantes}$ é o grafo das VLEs relevantes).

Saída: Ø

início

```
// Visita os nós-fornecedores do nó-union ou do nó-outer-union
para i de 1 até no_union_outer.Obtem_Qtd_Nos_Fornecedores() faça
    no_atual ← no_union_outer.Obtem_No_Fornecedor(i);
    // Verifica se o nó-fornecedor representa um Processo Filtro (F). Caso positivo, a visita é repassada ao seu primeiro
    // e único nó-fornecedor, que é um Processo Outer Union.
    se no_atual.valor() = "PF" então
      no_atual ← no_atual.Obtem_No_Fornecedor(1);
    fim se;
    Visita_No(no_atual, Q, G_Relevantes);
fim para;
fim.
```

Algoritmo: Visita_No_Consulta

Entrada: no_consulta, Q, $G_{Relevantes}$ (no_consulta é o nó que será visitado, Q é uma consulta sobre uma visão XML (pode ser sobre a VM ou VLEs) e $G_{Relevantes}$ é o grafo das VLEs relevantes).

Saída: Ø

início

```
// Obtendo do grafo das VLEs relevantes o objeto da consulta local e a subconsulta XQuery gerada na fase de decomposição.
no_vle_principal ← G_Relevantes.Obtem_No_VLE(no_consulta.Obtem_VLE());
Q' ← no_vle_principal.Obtem_Consulta_Local();
Q_XQuery ← no_vle_principal.Obtem_Subconsulta_XQuery();

// Verifica se o nó-consulta possui algum nó-fornecedor. Caso exista, indica que o Processo-Consulta em questão receberá
// valores para o parâmetro de um filtro sobre a chave da VLE. Portanto, para que seja novamente gerada uma subconsulta
// com um filtro parametrizado sobre a chave da VLE consultada, a chave da VLE é marcada para entrar na expressão de
// seleção da subconsulta.
se no_consulta.Existe_No_Fornece_Param() então
    // Se o nó-fornecedor é um nó-consulta, desconsidera o objeto da consulta local e gera-o novamente com base nas proprie-
    // dades que ainda estão sendo buscadas (caso citado no algoritmo Visita_No_Juncao_Multipla).
    se no_consulta.Obtem_No_Fornecedor(1).ehConsulta() então
        // Como Q pode ser avaliada várias vezes, limpa as marcações de propriedades suportadas pelas VLEs.
        Q.Limpa_Propriedades_Suportadas();
        Compara_Tipos(Q.Obtem_Tipo(), no_consulta.Obtem_VLE().Obtem_Tipo());
        Q' ← Q.Gera_Consulta_Local();
    fim se;
    Q'.Marca_Chave_VLE_Expr_Selecao();
    Q_XQuery ← Q'.Gera_Consulta_XQuery();
fim se;

// Gerando a subconsulta sobre a VLE do nó-consulta visitado e armazenando-a como atributo do nó-consulta.
no_consulta.Atribui_Subconsulta_XQuery(Q_XQuery);

// Avalia o grafo das VLEs relevantes e, se existirem VLEs Secundárias ligadas à VLE do nó-visitado, inclui os nós que
// representam os processos responsáveis por consultar e integrar as informações destas VLEs. Ao incluir nós responsáveis
// por consultar VLEs Secundárias, o algoritmo Visita_No_Consulta é chamado recursivamente, pois podem existir outras VLEs
// Secundárias relevantes ligadas às VLEs Secundárias dos nós que acabaram de ser incluídos.
se no_vle_principal.Obtem_Qtd_Nos_Filhos() > 0 faça
    // 1. Substituindo o nó-visitado por um nó-junção-múltipla
    no_pai ← no_consulta.Obtem_No_Pai();
    no_pjm ← Gera_No_PJM();
    no_pai.Substitui_No_Filho(no_consulta, no_pjm);
    // 2. Incluindo o nó-base, o qual representa o Processo-Consulta que fornece a coleção base das operações de junção.
    no_pjm.Adiciona_No_Base(no_consulta);
    // 3. Incluindo os nós responsáveis por consultar VLEs Secundárias ligadas à VLE do nó-consulta visitado.
    para cada i de 1 até no_vle_principal.Obtem_Qtd_Nos_Filhos() faça
        no_vle_secundaria ← no_vle_principal.Obtem_No_vle_secundaria(i);
        // Q'' é a consulta local que deve ser executada sobre a VLE Secundária.
        Q'' ← no_vle_secundaria.Obtem_Consulta_Local();
        // Gerando e incluindo o nó responsável por consultar a VLE Secundária ligada à VLE do nó-consulta visitado.
        // As informações da ligação e do tipo XML utilizado com base para a operação de junção são inseridas como atributos
        // do nó-junção-múltipla.
        [ℓ, T] ← G_Relevantes.Obtem_Atributo_Aresta(no_vle_principal, no_vle_secundaria);
        no_consulta_secundaria ← Gera_No_Consulta(no_vle_secundaria.Obtem_VLE());

        se no_vle_secundaria.Existe_Marcacao_Filtro() e
```

```
        não no_consulta.Existe_No_Fornece_Param() então
            // Incluindo o nó gerado à esquerda do nó-base.
            no_pjm.Adiciona_No_Filho_Esquerda_No_Base(no_consulta_secundaria, ℓ, T);
            no_consulta.Adiciona_No_Fornece_Param(no_consulta_secundaria);
            // Gerando e atribuindo novamente a subconsulta sobre a VLE do nó-consulta visitado. A nova subconsulta
            // conterá um filtro com parâmetro sobre a chave da propriedade prop envolvida na ligação. Para que a
            // subconsulta seja gerada com um filtro parametrizado sobre a chave da propriedade prop, a chave de prop é
            // marcada para entrar na expressão de seleção da subconsulta.
            prop ← ℓ.Obtem_Propriedade_Complexa_Envolvida_Ligacao();
            Q'.Marca_Chave_Propriedade_Expr_Selecao(prop);
            no_consulta.Atribui_Subconsulta_XQuery(Q'.Gera_Consulta_XQuery());
        senão
            no_pjm.Adiciona_No_Filho_Direita_No_Base(no_consulta_secundaria, ℓ, T);
            no_consulta_secundaria.Adiciona_No_Fornecedor(no_consulta);
        fim se;
        // Visitando recursivamente o nó-consulta incluído.
        Visita_No_Consulta(no_consulta_secundaria, Q'', G_Relevantes);
    fim para;
  fim se;
fim.
```

Algoritmo: Gera_Consulta_XQuery (método do objeto Consulta, que representa uma consulta que deverá ser executada sobre uma VLE)

Entrada: Ø (verifica as propriedades marcadas como suportadas na consulta Q' e as propriedades marcadas para compor sua expressão de seleção).

Saída: S (consulta XQuery sobre uma VLE, gerada a partir de Q').

```
início
  // Identificador do nível de aninhamento dos laços "for" (para propriedades multivaloradas).
  nivel ← 1;

  // Gera o cabeçalho da consulta, onde é definida a VLE consultada.
  S ←     " for $x1 in VIEW('" + Q'.Obtem_Nome_VLE() + "')/"
          + Q'.Obtem_Elemento_Primario_Visao();

  // Gera a expressão de seleção da consulta (cláusula "where").
  S ← S + Q'.Gera_Expressao_Selecao();
  S ← S + " return ";
  // Abre o elemento primário da cláusula "return"
  S ← S + "<" + Q'.Obtem_Elemento_Primario_Visao() + ">";

  // Gera trechos da consulta, correspondentes a cada elemento de Q'
  para cada p ∈ Q'.Obtem_Tipo() faça
    // Propriedades simples
    se p.ehSimples() então
      S ← S + " $x1/" + p.Obtem_Nome() + "," ;
    // Propriedades complexas
    senão
      // Armazena o caminho (path) do nível de aninhamento atual
      caminho ← " $x1/" + p.Obtem_Nome() + "," ;
      S ← S + Q'.Gera_Descricao_Prop_Complexa(p, caminho, nivel + 1);
    fim se
  fim para
```

```
// Fecha o elemento primário da cláusula "return"
S ← S + "</" + Q'.Obtem_Elemento_Primario_Visao() + ">";
retorna S; fim.
```

Algoritmo: Gera_Expressao_Selecao (método do objeto Consulta, que representa uma consulta que deverá ser executada sobre uma VLE)

Entrada: Ø (verifica as propriedades marcadas na consulta Q' que devem compor sua expressão de seleção).

Saída: S (trecho de consulta XQuery equivalente à expressão de seleção de Q').

```
início
  S ← "";
  // Verifica se a chave do tipo XML de Q' está marcada para compor a expressão de seleção. Caso positivo, a expressão de
  // seleção deve contemplar apenas um filtro com parâmetro sobre a chave da VLE consultada, ignorando as demais propriedades
  // marcadas para compor a expressão de seleção. Quando a chave da VLE não estiver marcada para compor a expressão de
  // seleção, verifica se existe chave do tipo de alguma propriedade complexa de Q' marcada para compor a expressão de seleção.
  // Caso positivo, gera a expressão de seleção com filtros com parâmetros sobre as chaves destas propriedades complexas.
  // Neste segundo caso, as demais propriedades marcadas para compor a expressão de seleção são também adicionadas à
  // expressão de seleção gerada. Os parâmetros dos filtros sobre propriedades serão posteriormente modificados por valores
  // extraídos de coleções XML.
  p ← Q'.Obtem_Chave_VLE_Marcada_Expressao_Selecao();
  se p ≠ nulo então
    S ← S + " where ";
    S ← S + "   $x1/ " + p.Obtem_Nome() + " in (PARAM1) ";
  senão
    se Q'.Existe_Chave_Prop_Marcada_Expressao_Selecao() então
      S ← S + " where ";
      i ← 1;
      para cada p ∈ Q'.Obtem_Chaves_Prop_Marcadas_Expressao_Selecao() faça
        // O método Obtem_Caminho(i) retorna o caminho de uma propriedade, a partir do elemento raiz do tipo da consulta,
        // removendo o caminho anterior à propriedade que está no nível i do caminho. Por exemplo, seja a propriedade
        // p = Aaa/Bbb/Ccc/Ddd/Eee, onde Aaa é o elemento raiz do tipo da consulta. O método p.Obtem_Caminho(1) retorna
        // 'Bbb/Ccc/Ddd', uma vez que a propriedade Bbb está no nível 1 do caminho, Aaa forma o caminho anterior à
        // propriedade Bbb e Eee é o nome da propriedade, obtido pelo método p.Obtem_Nome().
        S ← S + " $x1/ " + p.Obtem_Caminho(1) + "/" + p.Obtem_Nome()
                         + " in (PARAM" + i + ") ";
        i ← i + 1;
      fim para
    fim se
    se Q'.Existe_Prop_Marcada_Expressao_Selecao() então
      se S = "" então
        S ← S + " where ";
      fim se;
      para cada p ∈ Q'.Obtem_Prop_Marcadas_Expressao_Selecao() faça
        // As propriedades marcadas para compor a expressão de seleção possuem também como atributos:
        // (i) o valor a ser comparado com a propriedade e
        // (ii) o operador booleano que deverá ser considerado na comparação.
        S ← S + "   $x1/ " + p.Obtem_Caminho(1) + "\" + p.Obtem_Nome() + " "
                           + p.operadorExpSelecao() + " " + p.valorExpSelecao();
      fim para
    fim se  fim se  retorna S;
fim.
```

Algoritmo: Gera_Descricao_Prop_Complexa (método do objeto Consulta, que representa uma consulta que deverá ser executada sobre uma VLE)

Entrada: p, caminho, nivel (onde p é a propriedade complexa a qual se deseja gerar o trecho de consulta, caminho é o *path* do nível atual de aninhamento e nivel é o identificador do próximo nível de aninhamento dos laços "*for*").

Saída: S (trecho de consulta XQuery para a propriedade complexa p).

```
início
  // Verifica se a propriedade é monovalorada ou multivalorada, gerando laços "for" para aquelas propriedades multivaloradas e
  // chamando recursivamente o algoritmo Gera_Descricao_Prop_Complexa.
  se p.ehMonovalorada() então
    // Abre o elemento complexo e monovalorado
    S ← "<" + p.Obtem_Nome() + ">"
    // Gera trechos de consulta, correspondentes a cada subelemento de p
    para cada p' ∈ p.Obtem_Tipo() faça
      // Propriedades simples
      se p'.ehSimples() então
        S ← S + caminho + "/" + p'.Obtem_Nome() + "," ;
      // Propriedades complexas
      senão
        // Armazena o caminho (path) do nível de aninhamento atual
        caminho ← caminho + "/" + p'.Obtem_Nome() ;
        S ← S + Q'.Gera_Descricao_Prop_Complexa(p', caminho, nivel + 1);
      fim se
    fim para
    // Fecha o elemento complexo e monovalorado
    S ← S + "</" + p.Obtem_Nome() + ">"
  // propriedades multivalorada (necessário gerar laço for)
  senão
    // Gera o laço for
    S ←     " for $x" + nivel + " in " + caminho
    S ← S + " return " ;
    // Abre o elemento primário da cláusula "return"
    S ← S + "<" + p.Obtem_Nome() + ">"
    // Gera trechos da consulta, correspondentes a cada subelemento de p
    para cada p' ∈ p.Obtem_Tipo() faça
      // Propriedades simples
      se p'.ehSimples() então
        S ← S + " $x" + nivel + p'.Obtem_Nome() + "," ;
      // Propriedades complexas
      senão
        caminho ← " $x" + nivel + "/" + p'.Obtem_Nome()
        S ← S + Q'. Gera_Descricao_Prop_Complexa(p', caminho, nivel + 1);
      fim se
    fim para
    // Fecha o elemento primário da cláusula "return"
    S ← S + "</" + p.Obtem_Nome() + ">"
  fim se
  retorna S;
fim.
```

Referências Bibliográficas

[1] ABITEBOUL, S., BENJELLOUN, O., MILO, T. **Web Services and Data Integration**. *In:* Proceedings of the 3rd International Conference on Web Information Systems Engineering, p.3-6, December 12-14, IEEE Computer Society, 2002.

[2] ABITEBOUL, S., BUNEMAN, P., SUCIU, D. **Gerenciando Dados na Web**. 1a. Edição, Editora Campus, 1999.

[3] ADAMS, D. **Oracle® XML DB 10g Release 2 Developer's Guide**. 2005, http://www.oracle.com/technology/documentation/database10gR2.html, acessado em 07 de dezembro de 2007.

[4] ANDREWS, T., CURBERA, F., DHOLAKIA, H., GOLAND, Y., et al. **Business Process Execution Language for Web Services version 1.1**. Technical report, 2003, http://www.ibm.com/developerworks/library/specification/ws-bpel/, acessa-do em 07 de dezembro de 2007.

[5] ARTEMIS. **The ARTEMIS Project.** http://islab.dico.unimi.it/artemis, acessado em 17 de Agosto de 2009.

[6] BARU, C., GUPTA, A., LUDASCHER, B., et al. **XML-Based Information Mediation with MIX.** *In*: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Datam Philadelphia, PA, USA, 1999.

[7] BATINI, C., LENZERINI, M., NAVATHE, S. **A Comparative Analysis of Methodologies for Database Schema Integration.** *In*: ACM Computing Surveys 18, p. 323–364, 1986.

[8] BENEVENTANO, D., BERGAMASCHI, S., GUERRA, F., VINCINI, M. **Synthesizing an Integrated Ontology.** *In*: IEEE Internet Computing 7, p. 42-51, 2003.

[9] BLEIHOLDER, J., NAUMANN, F. **Conflict handling strategies in an integrated information system**. *In*: Proceedings of the International Workshop on Information Integration on the Web (IIWeb), Edinburgh, UK, 2006.

[10] BUSSE, S., KUTSCHE, R., LESER, U. et al. **Federated Information Systems: Concepts, Terminology and Architectures.** 1999.

[11] CASTANO, S., FERRARA, A. **Ontological Representation of Heterogeneous Datasources Integration Knowledge for the Semantic Web.** *In*: IASTED-ISC-03, Salzburg, Austria.. ACTA Press, p. 412-417, 2003.

[12] COSTA, T. **O Gerenciador de Consultas de um Sistema de Integração de Dados.**

Dissertação de Mestrado, Universidade Federal de Pernambuco, Recife, Pernambuco, Brasil, 2005.

[13] DOMENIG, R., DITTRICH, K. **An Overview and Classification of Mediated Query Systems.** ACM SIGMOD Record, vol. 28, n. 3, 1999.

[14] DRAGUT, E., LAWRENCE, R. **Composing Mappings Between Schemas Using a Reference Ontology**. *In*: Lecture Notes in Computer Science, Vol. 3290, p. 783-800, 2004.

[15] DRAPER, D., HALEVY, A. Y., WELD, D. S. **The Nimble Integration Engine**. *In:* Proceedings of ACM SIGMOD, Santa Barbara, California, USA, 21-24, 2001.

[16] EISENBERG, A., MELTON, J., KULKARNI, K., MICHELS, J.E., ZEMKE, F. **SQL:2003 has been published.** *In*: ACM SIGMOD Record, v. 33, n. 1 (Março 2004), COLUMN: Standards, pp. 119–126, 2004.

[17] ELMASRI, R., NAVATHE, S. B. **Fundamentals of Database Systems**, 5th. edition, Addison-Wesley, 2006.

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

[19] FIGUEIREDO, G. C. **Processamento de Consultas sobre Bases de Dados XML Distribuídas**. Dissertação de Mestrado, COPPE/UFRJ, Rio de Janeiro, Brasil, 2007.

[20] FUXMAN, A., HERNANDEZ, M. A., HO, H., et al. "**Nested mappings: schema mapping reloaded**". *In:* Proceedings of the 32nd international conference on Very Large Data Bases, Seoul, Korea, pp. 67-78, 2006.

[21] GARCIA-MOLINA, H., PAPAKONSTANTINOU, Y., QUASS, D., RAJARAMAN, A., SAGIV, Y., et al. **The TSIMMIS Approach to Mediation: Data Models and Languages.** *In*: Journal of Intelligent Information Systems, vol. 8, p. 117-132, 1997.

[22] GARDARIN, G., MENSCH, et al. **Integrating Heterogeneous Data Sources with XML and XQuery**. *In:* Proceedings of the 13th International Workshop on Database and Expert Systems Applications, pp. 839-846, 2002.

[23] HERNANDEZ, M., MILLER, A., RENEE, J. , HAAS, L. **Clio: A Semi-Automatic Tool for Schema Mapping.** ACM SIGMOD Int. Conf. on the Management of Data, 2001.

[24] HIROTO, K. KENJI, H. MIYAZAKI, J. UEMURA, S. **Efficient Query Processing for Large XML Data in Distributed Environments**. 21st International Conference on Advanced Networking and Applications, pp. 317-322, 2007.

[25] HURSON, A., BRIGHT, M., PAKZAD, S. **Multidatabase Systems: An Advanced Solution for Global Information Sharing**. Los Alamitos, CA, IEEE Computer Society Press, 1994.

[26] JAGADISH, H. V., LAKSHMANAN, L. V. S., SRIVASTAVA, D., et al. **TAX: A Tree Algebra for XML**. *In:* Database Programming Languages, 8th International Workshop, DBPL, pp. 149-164, 2001.

[27] LEE, K., MIN, J., et al. **A Design and Implementation of XML-Based Mediation Framework (XMF) for Integration of Internet Information Resources**. *In:* Proceedings of the 35th Annual Hawaii International Conference on System Sciences, v. 7, pp. 202-211, 2002.

[28] LEHTI, P., FANKHAUSER, P. **XML Data Integration with OWL: Experiences and Challenges.** *In*: 2004 Symposium on Applications and the Internet, p. 160–170, 2004.

[29] LEMOS, F. C. **Usando Assertivas de Correspondência para Especificação e Geração de Visões XML de Aplicações Web**. Dissertação de Mestrado, Universidade Federal do Ceará, Fortaleza, Ceará, Brasil, 2007.

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

[31] MANOLESCU, I., FLORESCU, D., KOSSMAN, D. **Answering XML queries over heterogeneous data sources.** *In:* VLDB, pp. 241-250, 2001.

[32] ÖZSU, M. T., VALDURIEZ, P. **Principles of Distributed Database Systems**, 2nd edition. Prentice Hall, 1999.

[33] PAN, A., RAPOSO, J., ALVAREZ, M., MONTOTO, et al. **The DENODO Data Integration Platform.** *In*: VLDB, 2002.

[34] PAPARIZOS, S., WU, Y., LAKSHMANAN, L. V. S. **Tree Logical Classes for Efficient Evaluation of XQuery**. *In*: Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, pp. 71-82, 2004.

[35] PARENT, C., SPACCAPIETRA, S. **Issues and Approaches of Database Integration.** *In*: CACM, 41(5), p. 166-178, 1998.

[36] SANTOS, L. **XML Publisher: Um Framework para Publicar Dados Objeto Relacionais em XML.** Dissertação de Mestrado, Universidade Federal do Ceará, Fortaleza, Ceará, Brasil, 2004.

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

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

[39] SHETH, A., LARSON, J. **Federated Database Systems for Managing Distributed, Heterogenous and Autonomous Databases.** *In*: ACM Computing Surveys 22, 1990, p. 183–236.

[40] TEIXEIRA, M. **Deegree Feature Publisher: Manual do Usuário.** 2004. http://www.lia.ufc.br/~teixeira/dfp, acessado em 07 de dezembro de 2007.

[41] TIMBER. **The TIMBER Project**. http://www.eecs.umich.edu/db/timber, acessado em 17 de Agosto de 2009.

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

[43] VIDAL, V., LÓSCIO, B. **Solving the Problem of Semantic Heterogeneity in Defining Mediator Update Translators.** *In*: Proc. of 18th Intern. Conf. on Conceptual Modeling, Paris, France, p.293-308, 1999.

[44] VIDAL, V., SANTOS, L, LEMOS, F. **Geração Automática de Visões de Objeto de Dados Relacionais**. 20th Brazilian Symposium on Databases, Uberlândia, Brazil, 2005.

[45] VIDAL, V., TEIXEIRA, M., FEITOSA, F. **Towards Automatic FeatureType Publication.** *In*: Proceedings of VI Brazilian Symposium on GeoInformatics. Campos do Jordão, Brazil, 2004.

[46] VIDAL, V., VILAS BOAS, R. **A Top-Down Approach for XML Schema Matching.** 17th Brazilian Symposium on Databases. Gramado, Brazil, 2002.

[47] W3C. World Wide Web Consortium. http://www.w3.org..

[48] WIEDERHOLD, G. **Mediators in the architecture of future information systems.** *In*: IEEE Computer, p. 38-49, 1992.

[49] YU, C., POPA, L. **Constraint-Based XML Query Rewriting For Data Integration.** *In:* SIGMOD Conference, p. 371-382, 2004.

[50] ZAMBOULIS, L., POULOVASSILIS, A. **Using AutoMed for XML Data Transformation and Integration.** *In*: Proceedings of DIWeb'04, CAiSE Workshop Proceedings, Volume 3, p. 58-69, 2004.

[51] ZIEGLER, P., DITTRICH, K. R. **Three Decades of Data Integration - All Problems Solved?**. *In*: Proceedings of 18th IFIP World Computer Congress (WCC 2004), Volume 12, Building the Information Society, pp. 3-12, 2004.