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

Dissertação de Mestrado

Acesso a Dados a partir de Ontologias Utilizando Mapeamentos Heterogêneos e Programação em Lógica

FERNANDA LÍGIA RODRIGUES LOPES

FORTALEZA/CE
Janeiro de 2011

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

Acesso a Dados a partir de Ontologias Utilizando Mapeamentos Heterogêneos e Programação em Lógica

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

Orientadora: Profª. Drª. Bernadette Farias Lóscio

FERNANDA LÍGIA RODRIGUES LOPES

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

Acesso a Dados a partir de Ontologias Utilizando Mapeamentos Heterogêneos e Programação em Lógica

Fernanda Lígia Rodrigues Lopes

Orientadora: Profª. Drª. Bernadette Farias Lóscio

Aprovada em __/__/_____

BANCA EXAMINADORA

Profª. Drª. Bernadette Farias Lóscio (Orientadora)
Universidade Federal de Pernambuco - UFPE

Profª. Drª. Maria Cláudia Reis Cavalcanti
Instituto Militar de Engenharia – IME

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

Prof. Dr. João Fernando Lima Alcântara
Universidade Federal do Ceará – UFC

AGRADECIMENTOS

Muitas pessoas foram importantes para que eu pudesse chegar até aqui. A cada uma delas, eu gostaria de deixar registrado meu sincero agradecimento.

Antes de tudo, agradeço a Deus, que me dá saúde e forças, o que me permite buscar o restante.

Agradeço à minha família, que é o meu alicerce. Obrigada por apoiarem as minhas escolhas, acreditarem em mim e se orgulharem a cada nova vitória. À minha mãe, Elisabete, que está sempre ao meu lado, oferencedo seu amor, atenção e suporte nos momentos necessários. Sem ela, eu não seria nada do que sou hoje. Ao meu pai, Ribamar, por seu amor e preocupação. Aos meus irmãos, Amanda e Clébio, meus grandes amigos, pelo companherismo, pelas implicâncias e por compartilharem comigo tantos momentos importantes. A todos os demais familiares que sempre torcem por mim.

À minha orientadora, professora Bernadette Farias Lóscio, com quem eu pude trabalhar e conviver durante estes anos. Agradeço por todo o aprendizado. Pela orientação presente, enorme paciência, dedicação, persistência, sugestões, incentivos, confiança e amizade. Com a professora Bernadette, eu aprendi muitas lições, tanto acadêmicas quanto sobre a postura de alguém que está sempre disposta a crescer, a lutar, a debater, a ouvir e a ensinar. Berna, muito obrigada por tudo! Eu tenho muito respeito pela pessoa e profissional que você é.

Aos professores Maria Cláudia Cavalcanti, João Fernando Lima Alcântara e José Antônio Fernandes de Macêdo por terem aceitado o convite de participar da banca examinadora deste trabalho. Agradeço ainda à professora Damires Fernandes que, infelizmente, não pôde estar presente na banca, mas que sempre foi muito atenciosa com questões relacionadas a este trabalho.

Ao professor João Fernando e ao Francicléber pelos esclarecimentos e troca de ideias sobre assuntos de lógica.

À minha amiga Eveline Sacramento, que foi uma grande parceira durante a realização de boa parte deste mestrado. Agradeço por nossas discussões enriquecedoras, pelo trabalho duro, pelos conselhos e incentivos. Eles foram muito importantes para que eu pudesse seguir em frente. Desejo, de maneira muito sincera, que a sua jornada seja repleta de sucesso e realizações.

Aos meus queridos amigos (grupo RTL) Hélio, Danusa e Bia, que estiveram presentes em tantos momentos durante estes anos. Foi muito bom conviver com vocês! Agradeço particularmente ao Hélio, que foi um companheiro de trabalho em algumas questões de implementação.

Aos amigos integrantes do grupo Arida, com os quais eu convivi, em algum momento: Fernando Lemos, Fernando Wagner, Ângela, Juliana, Bruno, Clayton, Lívia, Ticiana, Ticianne, Luiz Aires, João Carlos, Igo, André e Diego Victor.

Aos meus amigos Viviane, Paulo de Tarso, Katarina, Illana, Adriana e Bibi que, apesar da distância, uma hora ou outra se fizeram presentes com a sua amizade e troca de experiências pessoais e profisisonais.

A todos os professores e funcionários do MDCC que me ajudaram, de alguma forma, durante este caminho. Agradeço também ao secretário José Orley por sempre auxiliar nas pendências que surgiram.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e à Fundação Cearense de Apoio ao Desenvolvimento Científico e Tecnológico (Funcap) pelo auxílio financeiro que possibilitou a realização deste trabalho.

O desenvolvimento da capacidade geral de pensamento e livre-arbítrio sempre deveria ser colocado em primeiro lugar, e não a aquisição de conhecimento especializado. Se uma pessoa domina o fundamental no seu campo de estudo e aprendeu a pensar e a trabalhar livremente, ela certamente encontrará o seu caminho e será mais capaz de adaptar-se ao progresso e às mudanças.

(Albert Einstein)

Se o conhecimento nos faz descobrir problemas, não é a ignorância que os resolverá.

(Isaac Asimov)

RESUMO

Em várias áreas, tais como Integração de Dados e *Web* Semântica, ontologias têm sido adotadas para descrever formalmente a semântica das fontes de dados, com o intuito de facilitar a descoberta e a recuperação de informações. Dentro desse contexto, o Acesso a Dados Baseado em Ontologias (*Ontology-Based Data Access* - OBDA*)* é um problema decorrente da necessidade de acessar tais fontes a partir das ontologias que representam seus modelos conceituais. Dentre as principais características do OBDA, destacamos a independência entre as ontologias e a camada de dados e a possibilidade de responder a consultas que sejam mais expressivas que às geralmente realizadas utilizando Lógica Descritiva. Neste trabalho, especificamos um ambiente de OBDA no qual este problema é dividido em uma série de passos que podem ser tratados de maneira independente. Dentre cada um destes passos especificados, nossa principal contribuição reside na definição e implementação de um processo para reescrita de consultas entre ontologias estruturalmente distintas. Em nossa abordagem de reescrita, manipulamos a consulta de entrada combinando a semântica e a expressividade da linguagem SPARQL com um método baseado em noções de Programação em Lógica, uma vez que utilizamos mapeamentos heterogêneos expressos através de regras. Além disso, tratamos aspectos referentes às diferenças estruturais entre as ontologias, possibilitamos que partes da consulta reescrita possam ser descartadas durante o processo, caso seja constatado que tais partes seriam desnecessárias, e permitimos que os resultados sejam reestruturados e apresentados conforme a ontologia alvo. Por fim, é válido destacar que, embora a solução apresentada tenha como foco duas ontologias, esta pode ser estendida para considerar aspectos específicos de distribuição.

ABSTRACT

Ontologies have been used in different areas, including Data Integration and Semantic Web, to provide formal descriptions to data sources as well as to associate semantics to them and to make information easier to discover and to recover. In this context, one of the most relevant issues is the *Ontology-Based Data Access* – OBDA, which is the problem of accessing one or more data sources by means of a conceptual representation expressed in terms of an ontology. The independence between the ontology layer and the data layer, and the ability of answering more expressive queries than the ones defined using description logics are some of the main distinguished issues of the ODBA. In this work, we specify an environment for OBDA, which deals with this problem considering a set of independent tasks. Our main contribution concerns the definition and implementation of a query rewriting process between ontologies structurally heterogeneous. In the proposed query rewriting approach, we combine the semantics and expressiveness of SPARQL with logic programming and we adopt a rule-based formalism to represent mappings between ontologies. We also deal with some relevant questions, including: the structural heterogeneity, the prune of irrelevant parts of the rewritten query and the representation of query results according to the target ontology. It is important to note that, although in this work we discuss the use of the proposed solution considering just two ontologies, it can also be extended and applied for data distributions cenarios with multiple ontologies.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1
Introdução

1.1 Motivação e Caracterização do Problema

Em várias áreas, tais como Integração de dados e *Web* Semântica, ontologias têm sido utilizadas para descrever formalmente a semântica das fontes de dados, com o intuito de facilitar a descoberta das fontes que provêem as informações desejadas [Klien 2008; Lutz 2006], bem como a recuperação apropriada de tais informações. Além disso, ontologias têm sido ainda adotadas para especificar o esquema de mediação em sistemas de integração de dados [Calvanese et al. 2008].

Nestes casos, em geral, as ontologias descrevem o domínio de interesse e/ou representam o modelo conceitual dos dados de uma aplicação, os quais podem estar disponíveis em diversos formatos, como por exemplo: (i) no modelo relacional, visto que SGBDs relacionais são tecnologias de armazenamento consagradas; (ii) no formato XML que, devido à sua flexibilidade, tornou-se um padrão para representação e troca de dados na *web*; (iii) no modelo RDF, para algumas aplicações da *Web* Semântica.

Nesse contexto, a abstração a respeito dos detalhes estruturais de armazenamento e representação dos dados é possível através da criação de mapeamentos entre os elementos dos esquemas das fontes de dados e os conceitos nas ontologias. Uma vez estabelecidos estes mapeamentos, uma questão relevante consiste em determinar como utilizá-los, com o intuito de prover meios para que a informação disponível nas fontes de dados possa ser acessada a partir de sua representação semântica, ou seja, através das ontologias. Visando exemplificar um tipo de aplicação que lida com este problema, considere o cenário apresentado na Figura 1.1.

Esta figura ilustra a arquitetura de três camadas [Wache et al. 2001; Sacramento et al. 2010b] de um *framework* para integração de dados baseado em ontologias [Vidal et al. 2009], onde: (i) a ontologia de domínio (OD), além de representar o esquema de mediação, fornece um vocabulário de referência com os termos básicos de um domínio; (ii) a ontologia de aplicação (OA) descreve a semântica da fonte de dados a qual está associada, considerando o vocabulário e a estrutura da OD. Dessa forma, temos que as OAs são normalizadas de acordo com a OD, representando, portanto, um subconjunto da OD; (iii) os mapeamentos de mediação especificam o relacionamento entre as

ontologias de aplicação (OAs) e a ontologia de domínio (OD); (iv) os mapeamentos locais são utilizados para relacionar cada OA à sua respectiva base de dados.

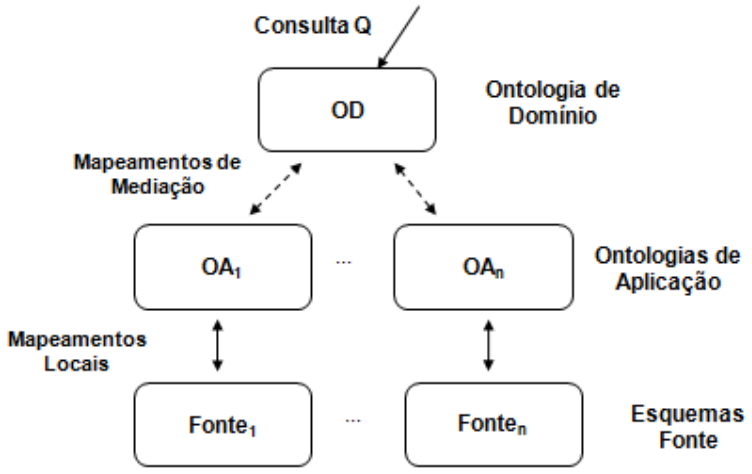


Figura 1.1. Um Framework Baseado em Ontologias para Integração de Dados Geográficos [Vidal et. al 2009]

Conforme abordado em [Sacramento et al.2010a], a utilização de uma arquitetura de três camadas facilita tanto a definição dos mapeamentos de mediação quanto o processo de reescrita de consultas para as várias fontes de dados. Nesta arquitetura, como as OAs correspondem a um subconjunto da OD, é possível definir mapeamentos homogêneos [Ghidini and Serafini 2006] no nível de mediação, enquanto os mapeamentos locais tornam-se mais complexos, uma vez que as ontologias de aplicação não refletem, necessariamente, a estrutura da fonte de dados. Com isso, torna-se necessário lidar com diferentes estruturas entre os dois modelos, o que demanda a utilização de mapeamentos heterogêneos. É válido destacar que, neste trabalho, esta classificação sobre o tipo do mapeamento é determinada com respeito às entidades envolvidas neste. Isto significa que um mapeamento é *heterogêneo* [Ghidini and Serafini 2006] se ele associa entidades de naturezas distintas, como por exemplo, uma classe em uma ontologia a uma propriedade em outra ontologia. Caso contrário, ele pode ser considerado *homogêneo*. Em geral, mapeamentos homogêneos podem ser representados através de associações mais diretas entre os termos envolvidos.

Ainda no cenário da Figura 1.1, é possível observar que os dados armazenados nas fontes devem ser acessados a partir de consultas realizadas sobre as ontologias. De maneira mais específica, consultas sobre as OAs devem ser reescritas em termos de suas respectivas fontes de dados, através da utilização de mapeamentos. Uma possível maneira de lidar com esta questão consiste no desenvolvimento de serviços que sejam

capazes de receber as consultas sobre as OAs e realizar todos os passos necessários para a correta execução desta consulta sobre as fontes, reestruturando os resultados conforme os conceitos e os relacionamentos expressos na ontologia sobre a qual a consulta foi submetida.

De modo geral, o problema descrito pode ser encontrado em aplicações nas quais seja necessário acessar dados a partir de ontologias tidas como modelos conceituais. Cenários típicos para este tipo de aplicação são as áreas de Integração de Informações e a *Web* Semântica [Calvanese et al. 2009]. Segundo Calvanese [Calvanese et al. 2009; Poggi et al. 2008], este tipo de acesso pode ser denominado de *Acesso a Dados baseado em Ontologias* (*Ontology-Based Data Access* - OBDA*)* e tem como uma de suas principais características a independência entre as ontologias e a camada de dados. Além disso, um segundo aspecto fundamental em OBDA consiste na possibilidade de responder a consultas que sejam mais expressivas que às geralmente realizadas utilizando lógica descritiva. Como veremos posteriormente, estes dois aspectos são considerados na abordagem aqui apresentada.

Em [Poggi et al. 2008; Calvanese et al. 2009], os autores aplicam a definição de OBDA para dados armazenados em SGBDs relacionais. Em nossa abordagem, nós adaptamos e estendemos este conceito, nos abstraindo de um modelo de dados específico, a fim de obter uma solução mais genérica e modular. Para isto, dividimos o problema em etapas que podem ser tratadas de maneira independente. Tais etapas são: (i) tradução sintática do esquema da fonte de dados para uma ontologia, gerando o que é chamado de ontologia local; (ii) criação dos mapeamentos entre a ontologia local e a ontologia de domínio; (iii) reescrita da consulta entre estas ontologias, o que facilita a posterior tradução desta consulta para a linguagem da respectiva base de dados. Observe que a etapa (i) é semelhante à idéia dos esquemas de exportação dos ambientes de integração de dados. Em particular, neste trabalho, focamos no problema (iii), propondo, implementando e validando um processo para resolvê-lo.

Neste ponto, é importante salientar que, ao considerarmos tanto o esquema fonte quanto o esquema alvo como ontologias, apresentamos uma solução para a reescrita que não se aplica somente à OBDA. Tal solução é válida para outros cenários, nos quais seja necessário reescrever consultas entre ontologias. Como exemplo, podemos citar os sistemas baseados em ontologias que adotam arquiteturas *peer-to-peer* para integração de informações [Pires 2009; Adjiman et al. 2007].

Note que estamos lidando com um problema geral de *reescrita de consultas entre ontologias*, no qual, dada a natureza heterogênea dos cenários onde este problema aparece, não podemos esperar que as ontologias sejam semelhantes, isto é, apresentem estruturas homogêneas. Este problema é apresentado na Figura 1.2 e descrito a seguir. Considere uma ontologia alvo O_T, uma ontologia fonte O_S e um conjunto de mapeamentos M_{ST}, estabelecidos entre estas ontologias. Considere ainda que O_T é a visão fornecida aos usuários e aplicações, ou seja, é a ontologia a qual estes tem acesso. Seja Q_T uma consulta formulada de acordo com O_T. Para que Q_T possa ser também executada sobre O_S, é necessário reescrevê-la em uma nova consulta Q_S, expressa em termos O_S. Tal consulta reescrita é derivada a partir de um processo que explora o conjunto de mapeamentos M_{ST}. É importante observar que a ontologia O_T tanto pode representar apenas uma visão, como nos cenários de OBDA, quanto pode possuir instâncias.

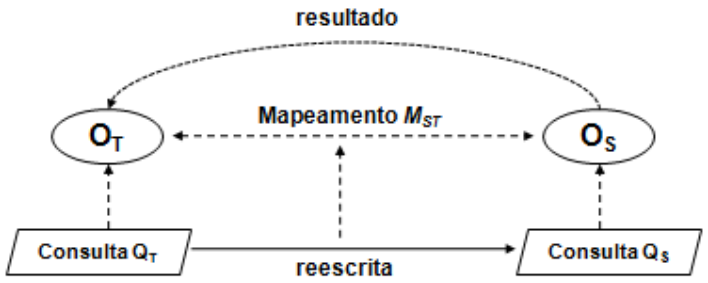


Figura 1.2. Reescrita de consultas entre duas ontologias

Após a geração e execução da consulta Q_S, há ainda a necessidade de retornar os dados na representação de O_T, por um dos seguintes motivos: (i) como mencionado, O_T é a visão fornecida aos usuários e aplicações; (ii) os resultados das consultas serão posteriormente unificados com as instâncias presentes em O_T, caso estas existam, e/ou com instâncias oriundas de outras ontologias, que também devem estar de acordo com O_T.

1.2 Objetivos e Contribuições da Dissertação

Este trabalho tem como objetivo geral a especificação de um ambiente que possibilite que fontes de dados possam ser acessadas a partir de uma interface baseada em ontologias, considerando que ontologias e fontes de dados possuem estruturas e níveis de abstração distintos. Para isto, cada um dos pontos (i), (ii) e (iii) citados anteriormente foram abordados, com ênfase no ponto (iii), que consiste na principal contribuição desta

dissertação. Com relação ao ponto (i), realizamos um levantamento e análise dos trabalhos existentes, visando identificar aqueles mais propícios a serem utilizados no ambiente proposto. No que se refere ao aspecto (ii), ao considerarmos que as ontologias podem apresentar diferenças estruturais, houve a necessidade de representar a reestruturação de objetos, através da definição de *mapeamentos heterogêneos* entre as entidades de ambas as ontologias. Sendo assim, empregamos um método [Sacramento et al. 2010a] para geração destes mapeamentos, os quais são expressos através de um formalismo baseado em regras. No que tange ao ponto (iii), propomos, implementamos e validamos um processo para reescrita de consultas entre ontologias. É válido destacar que, embora a solução apresentada tenha como foco duas ontologias, esta pode ser estendida para considerar aspectos específicos de distribuição, conforme necessário.

Portanto, temos que as principais contribuições desta dissertação são:

- Especificação de um ambiente para acesso a dados baseado em ontologias, tendo como núcleo um processo para reescrita de consultas entre duas ontologias. Para esta especificação, foram analisados trabalhos existentes que pudessem ser aplicados, bem como foram caracterizados os mapeamentos utilizados durante a reescrita.
- Definição de um processo para reescrita de consultas entre ontologias estruturalmente distintas. Em nossa abordagem de reescrita, propomos um modelo para representação de uma consulta SPARQL e apresentamos um algoritmo que, utilizando-se deste modelo, reescreve uma consulta definida sobre uma ontologia alvo em uma consulta sobre uma ontologia fonte. As principais características do processo de reescrita proposto residem nos seguintes aspectos: (i) manipulamos a consulta de entrada combinando a semântica e a expressividade dos construtores de SPARQL com um método baseado em noções de programação em lógica, adaptando estas noções para o nosso contexto; (ii) tratamos aspectos referentes às diferenças estruturais entre as ontologias, fazendo uso de mapeamentos heterogêneos e considerando a utilização de predicados de comparação (>, <, =, etc.) tanto nos mapeamentos, quanto nas consultas, sendo estes predicados analisados durante o processo de reescrita; (iii) possibilitamos que partes da consulta reescrita possam ser descartadas, caso seja constatado que tais partes seriam redundantes ou retornariam vazio; (iv) permitimos que os resultados sejam reestruturados e apresentados conforme a ontologia alvo, na qual a consulta foi submetida.

- Desenvolvimento de um protótipo com funcionalidades para a definição da consulta, reescrita, apresentação dos resultados e manipulação dos mapeamentos. Este protótipo foi utilizado para a validação experimental.

1.3 Organização da Dissertação

Esta dissertação está organizada da seguinte forma:

- O Capítulo 2 contém a fundamentação teórica, apresentando uma revisão sobre os tópicos relevantes para o desenvolvimento desta dissertação, bem como os principais trabalhos relacionados.
- O Capítulo 3 apresenta a especificação do ambiente para acesso a dados baseado em ontologias proposto neste trabalho. Dentro do contexto deste ambiente, é explanado o processo de geração e caracterização dos mapeamentos utilizados.
- O Capítulo 4 descreve a abordagem proposta para reescrita de consultas entre ontologias, detalhando cada uma de suas etapas.
- O Capítulo 5 apresenta o protótipo desenvolvido, discorrendo sobre suas principais funcionalidades, aspectos de implementação e resultados experimentais.
- Finalmente, o Capítulo 6 expõe as conclusões e perspectivas de trabalhos futuros.

CAPÍTULO 2
Fundamentação Teórica

Neste capítulo, apresentamos os conceitos básicos utilizados nesta dissertação, assim como uma revisão bibliográfica acerca dos assuntos relacionados à nossa abordagem. Para isto, na Seção 2.1, é discutido o problema de Acesso a Dados Baseado em Ontologias, sendo levantadas questões relativas aos seguintes aspectos: (i) tradução entre ontologias e outros modelos de dados, (ii) mapeamento entre ontologias e (iii) reescrita de consultas. Em seguida, a Seção 2.2 discorre sobre as principais abordagens para reescrita de consultas com ontologias. Tais abordagens estão entre as mais próximas do nosso trabalho. Por fim, a Seção 2.3 apresenta uma análise comparativa entre os trabalhos relacionados, incluindo a nossa estratégia.

2.1 Acesso a Dados Baseado em Ontologias

Acesso a Dados Baseado em Ontologias (*Ontology-based Data Acess* – OBDA) consiste no problema de acessar uma ou mais fontes de dados existentes através de uma representação conceitual, de alto nível e formal destas fontes, expressa sob a forma de ontologias. Exemplos de cenários nos quais este problema aparece com freqüência são os Sistemas de Integração de Dados e a *Web* Semântica. Tendo em vista que o denominador comum em todas estas aplicações é o acesso a um conjunto de dados através de uma ontologia, esta nomenclatura foi adotada na literatura [Poggi et al. 2008] e, portanto, será utilizada neste trabalho. A seguir, apontamos os principais aspectos deste problema, visando caracterizá-lo.

1. As fontes de dados devem existir de maneira independente, significando que estas não foram necessariamente desenvolvidas de acordo com a(s) ontologia(s) com as quais estão associadas. Portanto, não podemos fazer suposições a respeito da estrutura de ambos os modelos.

2. Ontologias e fontes de dados (i) apresentam níveis de abstração e expressividade diferentes e (ii) são expressas sob formalismos distintos. Enquanto ontologias devem ser descritas através de alguma linguagem lógica (Lógica Descritiva, por exemplo), de modo a capturar as construções normalmente utilizadas em

modelagem conceitual, fontes de dados podem estar representadas segundo o modelo relacional, XML, etc.

3. Um serviço fundamental em OBDA diz respeito a responder a consultas submetidas sobre as ontologias, onde: (i) um conjunto de mapeamentos deve ser considerado; (ii) estas consultas devem ser mais complexas que às permitidas em pesquisas envolvendo DL. Isto porque DL pode ser utilizada para capturar classes e relacionamentos na ontologia. Entretanto, como linguagem de consulta, ela é bastante limitada, principalmente quanto à realização de projeção e junção [Calvanese et al. 2009]. Além disso, DL não possui a flexibilidade dos construtores presentes nas linguagens de consultas, como por exemplo, em SPARQL.

Neste trabalho, nós estendemos e adaptamos a ideia apresentada em [Poggi et al. 2008], dividindo o problema de OBDA em atividades que podem ser tratadas de maneira independente. As seções subsequentes apresentam os principais conceitos e trabalhos existentes acerca das principais linhas de pesquisa relacionadas com nossa abordagem. Destacamos as seções voltadas para o problema de reescrita, que consiste no núcleo desta dissertação.

2.1.1 Tradução entre Ontologias e Modelo de dados Relacional e XML

Descrever fontes de dados através de ontologias constitui um dos principais alicerces para a concretização da *Web* Semântica, dada a vasta quantidade de conteúdo legado existente. Além disso, tal tarefa é de extrema importância para aplicações de Acesso e Integração de Dados baseado em Ontologias. Assim, tendo em vista a relevância desta atividade, muito trabalho tem sido realizado neste sentido. Em geral, estes trabalhos focam no desenvolvimento de tradutores, nos quais os construtores de uma fonte de dados são mapeados para construtores ontológicos, utilizando-se de diversas técnicas que podem ser manuais ou automáticas.

Nesta seção, apresentamos as principais características deste processo de tradução, bem como uma revisão bibliográfica dos trabalhos existentes, com o intuito de determinar aqueles que podem ser aplicados no ambiente proposto. Para isto, esta revisão bibliográfica está organizada da seguinte maneira:

(i) Inicialmente, apresentamos os trabalhos que traduzem fontes de dados relacionais e XML para ontologias. Nesta categoria, estes podem ser subdivididos da seguinte forma: aqueles que somente exportam o esquema; os que exportam o esquema e os dados; aqueles que exportam o esquema e geram mapeamentos simples entre os

dois modelos. Note que estamos interessados em abordagens nas quais a ontologia local seja gerada de maneira automática e simplificada, uma vez que trataremos da complexidade do problema de reescrita entre as duas ontologias (local e domínio).

(ii) A maioria dos trabalhos descritos em (i) lida apenas com o problema de tradução de modelos. No entanto, caso a ontologia não seja povoada com as instâncias do banco de dados (visão materializada), é necessário possibilitar que uma consulta SPARQL seja transformada em uma consulta SQL ou *XQuery* [Boag et al. 2007]. Para isso, expomos as principais abordagens para tradução entre estas linguagens de consulta, discorrendo sobre seus objetivos e como estas abordagens poderiam complementar os trabalhos vistos em (i).

Ao final de cada subseção, apresentamos nossas conclusões a respeito dos trabalhos aptos a serem instanciados em nosso ambiente.

Tradução entre Ontologias e o Modelo de Dados Relacional

De maneira geral, em todos os trabalhos analisados, o processo de tradução segue o mapeamento de construtores ilustrado na Tabela 2.1. A seguir, serão exibidas as características específicas destes trabalhos, bem como uma tabela (Tabela 2.2) que sumariza tais características.

Tabela 2.1. Tradução entre os construtores do modelo relacional e de uma ontologia OWL

Construtor no Esquema Relacional	Construtor na Ontologia (OWL)
Relação[1]	Classe
Atributo não pertencente à chave estrangeira	Propriedade de tipo de dado
Atributo chave estrangeira	Propriedade de objeto
Chave primária	Propriedade funcional ou inversa funcional
Chave primária e estrangeira juntas	Representada como herança
Restrição de UNIQUE	Funcional ou inversa funcional
Restrição de NOT NULL	Funcional ou cardinalidade mínima igual a 1

Astrova [Astrova and Kalja 2008] apresenta um protótipo para transformação automática de um esquema de banco de dados relacional em uma ontologia OWL *Full*, visando atualizar os dados para *Web* Semântica. Como entrada, a ferramenta recebe o SQL-DDL de geração do banco de dados e, então, os construtores SQL (incluindo as restrições) são mapeados para construtores em OWL, possibilitando a geração da

[1] Relações que possuem somente chaves estrangeiras não são transformadas em classes, pois tais relações não expressam entidades, mas sim relacionamentos.

ontologia. Durante o processo, as instâncias do banco de dados são utilizadas para povoar a ontologia.

Trinh [Trinh et al. 2006] propõe uma ferramenta, chamada *RDB2ONT*, que cria ontologias em OWL, representadas no modelo *Relational.OWL.* Este modelo é uma meta-ontologia que descreve os componentes de um esquema relacional. Por exemplo, nesta ontologia existem as classes "Tabela", "Coluna", "Chave Primária", etc. Dessa forma, os construtores do esquema do banco de dados são representados como instâncias na ontologia *Relational.OWL.*

O projeto *DataMaster* [Nyulas et al. 2007] é um *plugin* para o *Protegé* que possibilita a geração automática de uma ontologia a partir de um banco de dados relacional. O processo de geração é simples e direto, seguindo, basicamente, o mapeamento apresentado na Tabela 2.1. Além disso, as instâncias da base de dados são transformadas em instâncias na ontologia e o usuário pode selecionar quais construtores deseja traduzir.

Em [Sonia and Khan 2008], é apresentada uma técnica para transformação de modelos relacionais em ontologias, através de uma extensão do trabalho de engenharia reversa proposto em [Alhajj 2003], o qual extrai um diagrama EER (entidade-relacionamento) a partir de um banco de dados legado.

Já *Cerbah* [Cerbah 2008], procede com a transformação de acordo com a Tabela 2.1, mas considera ainda o conteúdo do banco de dados, caso o usuário deseje. Tal conteúdo é utilizado para gerar refinamentos nas hierarquias da ontologia obtida, incluindo informações adicionais.

Finalmente, *Lubyte* [Lubyte and Tessaris 2009] propõe um processo para extração de ontologias a partir de esquemas relacionais, através da análise das restrições presentes no banco de dados. A partir desta análise, é aplicado um processo de engenharia reversa que permite a geração da ontologia e de um conjunto de mapeamentos que conectam esta ontologia com seu respectivo esquema relacional. Além disso, os autores demonstram formalmente que a ontologia extraída reflete adequadamente a fonte de dados, de modo que não há perda de informação e nenhuma informação extra é adicionada.

Diante dos trabalhos analisados, e observando a Tabela 2.2, podemos fazer algumas comparações. Todas as abordagens, com exceção de [Sonia and Khan 2008] e [Lubyte and Tessaris 2009], povoam a ontologia diretamente com as instâncias do banco de dados. No caso de [Sonia and Khan 2008], a idéia é aplicar técnicas de

engenharia reversa na tentativa de resgatar o modelo conceitual original da fonte de dados e representá-lo através da ontologia gerada. Por este motivo, a única utilização das instâncias é no sentido de auxiliar na obtenção da cardinalidade dos relacionamentos. Por outro lado, *Lubyte* [Lubyte and Tessaris 2009] não materializa as instâncias na ontologia porque gera os respectivos mapeamentos simples entre esta e o esquema da fonte de dados.

Alguns trabalhos estão mais focados em extrair ontologias com hierarquias mais refinadas [Cerbah 2008], enquanto outros podem ser eliminados devido à necessidade de dispor do *script* SQL de geração do banco de dados [Astrova and Kalja 2008] ou de representar a ontologia no modelo *Relational.OWL* [Trinh et al. 2006]. Dentre as abordagens, a única que extrai a ontologia, gera os respectivos mapeamentos e valida a transformação de maneira teórica, além da experimental, é [Lubyte and Tessaris 2009]. Com isso, temos a garantia de que não há perda de informação durante o processo de tradução de modelos. Portanto, apontamos tal trabalho como o mais apropriado e completo, desde que não se deseje povoar as ontologias com todas as instâncias da fonte de dados. Caso contrário, [Nyulas et al. 2007] e [Cerbah 2008] podem ser utilizados.

Tabela 2.2. Resumo das abordagens para geração de ontologias a partir de bancos de dados relacionais

Abordagem	Objetivo	Entrada	Nível de Automação	Aspectos analisados do BD			Povoamento da Ontologia	Conexão da Ontologia com o BD	Validação
				Relações e atributos	Restrições	Dados			
Astrova [Astrova and Kalja 2008]	Migrar os dados para Web Semântica	SQL-DDL	Automático	Sim	Sim	Não	Direto com as instâncias do banco	Não mantém conexão	Experimental
RDB2ONT [Trinh et al. 2006]	Migrar os dados para Web Semântica e prover interoperabilidade entre aplicações com ontologias	Esquema do BD Relacional	Automático	Sim	Parcialmente	Não	Direto com as instâncias do banco	Não mantém conexão	Experimental
DataMaster [Nyulas et al. 2007]	Importar a estrutura de um BD relacional para uma ontologia OWL no ambiente *Protegé*	Esquema do BD Relacional	Automático	Sim	Parcialmente	Não	Direto com as instâncias do banco	Não mantém conexão	Experimental
Sonia et al [Sonia and Khan 2008]	Criação de ontologia local para um banco de dados	Banco de dados Relacional	Automático	Sim	Sim	Somente para auxiliar na determinação da cardinalidade dos relacionamentos	Não povoa	Não mantém conexão	Experimental
Cerbah [Cerbah 2008]	Gerar uma ontologia bem estruturadas a partir de um BD	Banco de dados Relacional	Automático ou Semi Automático	Sim	Parcialmente	Sim, caso o usuário deseje	Direto com as instâncias do banco	Não mantém conexão	Experimental
Lubyte [Lubyte and Tessaris 2009]	Extrair uma ontologia que reflita precisamente o esquema do banco de dados relacional	Esquema do BD Relacional	Automático	Sim	Sim	Não	Não povoa	Geração de mapeamentos simples estilo GAV	Teórica e Experimental

Tradução entre Ontologias e o Modelo de Dados XML

O modelo XML, além de possuir mais construtores que o modelo relacional, permite uma maior flexibilidade para a utilização e combinação destes construtores. Dessa forma, os trabalhos para tradução de esquemas XML para ontologias apresentam um padrão de transformação um pouco mais heterogêneo que as abordagens vistas anteriormente. Em geral, tais trabalhos se diferenciam com relação aos casos mais complexos, que abrangem grupos de elementos, sequências ou elementos anônimos. A Tabela 2.3 expõe os principais construtores mapeados, seguida de uma breve explanação sobre abordagens analisadas. Em seguida, é apresentada a Tabela 2.4, que sumariza as principais características destas abordagens.

Tabela 2.3. Tradução entre os construtores do modelo XML e de uma ontologia OWL

XML	Ontologia
Tipos Complexos (*Complex Type*)	Classe
Grupo de Elementos (*Group*)	Classe
Grupo de Atributos (*Attribute Group*)	Classe
Extensão ou Restrição em Tipos Complexos (*extension, restriction*)	Subclasse da classe correspondente ao tipo base
Elemento (*element*)	Propriedade de tipo de dados (elemento simples) ou propriedade de objeto (relacionando subelementos)
Atributo (*attribute*)	Propriedade de tipo de dado
Sequência (*sequence*)	Classe não nomeada - Interseção
Construtor de elemento Opcional (*choice*)	Classe não nomeada - União
Número máximo de Ocorrências (*maxOccurs*)	*maxCard*
Número mínimo de Ocorrências (*minOccurs*)	*minCard*

Garcia [Garcia and Celma 2005] propõe uma estratégia, chamada de *XSD2OWL*, onde um esquema XML é transformado em uma ontologia OWL, seguindo os casos apresentados na Tabela 2.3. Embora o processo apresentado seja genérico, a aplicação desta transformação tem como foco a integração e a recuperação de metadados multimídia, como por exemplo, informações do *framework* MPEG-7 [Manjunath et al. 2002], que é baseado em esquemas XML. Neste trabalho, além da geração da ontologia, os autores executam a importação dos dados XML através de um processo chamado de *XML2RDF*, onde estes dados são armazenados em um repositório RDF (*RDF Store*), juntamente com as ontologias. Com isso, eles mostram que a tarefa de integração e recuperação é facilitada por meio do uso de motores de inferência e classificação sobre o repositório gerado.

Em [Bohring and Auer 2005], é apresentada uma abordagem na qual uma ontologia OWL é gerada a partir de um documento XML, não havendo a necessidade de dispor do esquema XML, caso este não exista. Os autores derivam o esquema a partir dos dados e, então, procedem com a transformação conforme a Tabela 2.3. Finalmente, ilustram como converter os dados XML para instâncias OWL utilizando a linguagem XSLT [Clark 1999].

Já o processo *X2OWL* [Ghawi and Cullot 2009] recebe como entrada um esquema XML e gera uma ontologia OWL de acordo com a Tabela 2.3, tratando alguns casos adicionais, tais como a existência de elementos locais anônimos e tipos complexos mistos (que possuem texto e subelementos). Além disso, os autores não importam as instâncias para uma ontologia ou um repositório RDF. Eles optam por gerar, durante a transformação, um documento de mapeamento que descreve as correspondências entre as entidades da fonte XML e da ontologia local resultante. Estas correspondências são descritas por meio de expressões de caminho (*XPath* [Clark 1999]).

XS2OWL [Tsinaraki and Bikakis 2007] é um *framework* que também realiza a transformação de um esquema XML para uma ontologia OWL. Assim como [Ghawi and Cullot 2009], a ontologia gerada não é povoada com as instâncias do documento XML, mas são criados mapeamentos entre os dois modelos, sendo estes mapeamentos expressos em *XPath*. Este trabalho se diferencia dos demais por ser complementado por outros dois processos [Bikakis et al. 2009a], desenvolvidos pelos mesmos autores. O primeiro processo permite que os dados em RDF sejam transformados novamente em XML, caso seja necessário. Com isso, o trabalho possibilita a troca de informações entre aplicações que lidam com XML e RDF. Já o segundo processo, é responsável por gerar e manter mapeamentos entre os esquemas XML e as ontologias extraídas. Por fim, os autores propõem ainda um *framework* mais geral [Bikakis et al. 2009b], o qual permite que uma consulta sobre uma ontologia local seja realizada em SPARQL. Tal consulta é posteriormente traduzida para *XQuery*, utilizando os mapeamentos estabelecidos em *XPath*.

A respeito destas abordagens que realizam a tradução entre XML e OWL, os dois primeiros trabalhos são úteis quando houver a necessidade de armazenar todas as instâncias na ontologia local. Observe que o método descrito em [Garcia and Celma 2005] deve ser utilizado para povoar um RDF *Store* com tais instâncias.

Tabela 2.4. Resumo das abordagens para geração de ontologias a partir esquemas/dados XML

Abordagem	Objetivo	Entrada	Saída	Nível de Automação	Povoamento da Ontologia	Conexão da Ontologia com o BD	Validação
***XSD2OWL* e *XML2RDF* [Garcia and Celma 2005]**	Processo genérico aplicado ao domínio de metadados Multimídia, com intuito de facilitar a integração e recuperação destes metadados.	Esquema XML. Dados XML, caso se deseje importá-los	Ontologia OWL. Instâncias no repositório RDF.	Automático	Importa e armazena os dados XML como dados RDF, em um *RDF Store*.	Não mantém conexão	Experimental. Informações domínio de metadados Multimídia
Bohring [Bohring and Auer 2005]	Gerar a ontologia, com suas respectivas instâncias, quando não se dispõe do esquema XML da fonte de dados. Usa XSLT para importação dos dados em OWL.	Dados XML. Esquema, somente se este existir.	Ontologia OWL povoada com as instâncias do documento XML.	Automático	Transforma os dados XML em instâncias da ontologia OWL gerada. Para isso, utiliza XSLT.	Não mantém conexão	Experimental. Arquivos do *Citeseer*
***X2OWL* [Ghawi and Cullot 2009]**	Trata casos adicionais, específicos para elementos anônimos e mistos. Além disso, o gera o arquivo de correspondências.	Esquema XML.	Ontologia OWL. Documento com as Correspondências entre os esquemas.	Automático ou Semi-Automático (refinamentos)	Não povoa	Gera um documento com as correspondências (em *XPath*) entre as entidades da fonte XML e da ontologia gerada.	Ferramenta com um estudo de caso utilizado como exemplo
Framework *XS2OWL* [Tsinaraki and Bikakis 2007], complementado por [Bikakis et al. 2009a]	Prover funcionalidades para o processo completo de interoperabilidade entre XML e RDF: geração da ontologia local em OWL, bem como dos mapeamentos simples; transformação de RDF para XML e acesso aos dados XML a partir de uma consulta SPARQL.	Esquema XML.	Ontologia OWL. Mapeamentos entre os esquemas / Documento XML do RDF.	Automático ou Semi-Automático (refinamentos)	Não povoa	Gera: (i) mapeamentos (em *XPath*) entre o esquema XML e a ontologia; (ii) documento para permitir a transformação RDF-XML.	Experimental com esquemas e documentos diversos

Os demais trabalhos, ao contrário, geram documentos com mapeamentos entre os modelos, ambos expressos em *XPath*. Neste caso, a vantagem de utilizar *XS2OWL* reside no fato deste ser um framework completo, com uma validação experimental bem desenvolvida, ao passo que *X2OWL*, embora trate de construtores XML bem específicos, apresenta apenas um estudo de caso simples como forma de validação.

Tradução de SPARQL para SQL e *XQuery*

A maioria dos trabalhos que tratam do problema de traduzir consultas SPARQL para SQL tem como principal objetivo contribuir com projetos que visam à construção de repositórios RDF (RDF *Store*) no topo de SGBDs Relacionais. Estes projetos almejam gerenciar, de maneira eficiente, grandes quantidades de triplas RDF. Para isso, propõem armazenar fisicamente tais triplas em bancos de dados relacionais (na forma de tabelas com três atributos: sujeito, predicado e objeto), com o intuito de tirar proveito dos benefícios vinculados à indexação, gerenciamento, desempenho, etc. Por outro lado, para que os dados RDF possam consultados na interface de um RDF *Store*, o qual recebe uma consulta em SPARQL, é necessário prover uma tradução eficiente de SPARQL para SQL. É valido salientar que esta aplicação voltada para repositórios RDF está relacionada aos dados que já se encontram disponíveis neste modelo. No entanto, devemos considerar também as informações que estão nativamente armazenadas em SGBDs relacionais. Diante disso, podemos citar como uma motivação geral destes trabalhos, a necessidade de se prover interoperabilidade entre aplicações que armazenam seus dados no modelo relacional com aquelas que recebem consultas em SPARQL.

Dentre os principais trabalhos inseridos nesta categoria, Cyganiak [Cyganiak 2005] define uma álgebra relacional para SPARQL e descreve um conjunto de regras que estabelecem a equivalência entre esta álgebra e SQL. Li Ma [Ma et al. 2008] propõe um método que traduz uma consulta SPARQL em uma única consulta SQL aninhada. Neste método, cada padrão de nó é traduzido em uma subconsulta SQL. Além disso, os autores lidam com expressões de filtro, abordam questões de eficiência e fazem uma série de avaliações utilizando o SBGD DB2. Como limitações desta abordagem, podemos citar a falta de suporte a alguns operadores, como por exemplo, GRAPH, *datasets* nomeados e modificadores de solução. Chebotko [Chebotko et al. 2009] apresenta um algoritmo e demonstra que este preserva a semântica na tradução de SPARQL para SQL. Neste trabalho, os autores propõem que esta tradução seja baseada

no uso de *templates* SQL tanto para os padrões de tripla quanto para os operadores de junção, união, opção, filtro e seleção. Por fim, ainda desenvolvem simplificações que permitem a geração de consultas mais simples e eficientes. Já Elliot [Elliott et al. 2009] estende o trabalho apresentado em [Chebotko et al. 2009] mostrando que a geração de consultas muito aninhadas possui uma baixa escalabilidade para consultas que envolvem mais de 3 ou 4 padrões de triplas. Com isso, é proposto um método que implementa cada um dos operadores de SPARQL por meio do "aumento" de uma consulta SQL, e não somente através do aninhamento de consultas. Uma vez estabelecido este algoritmo, a tradução dos operadores não tratados em abordagens anteriores (GRAPH, *datasets* nomeados e modificadores de solução) é realizada. Os autores apresentam ainda uma série de resultados experimentais que validam a eficiência do método proposto.

Note que, embora estas abordagens não traduzam a consulta utilizando mapeamentos entre esquemas, elas focam em associar cada um dos construtores de SPARQL com os elementos de SQL. Por este motivo, estes métodos podem ser empregados para tradução das consultas, podendo ser incorporado a eles, a utilização dos mapeamentos locais.

No que se refere à tradução de SPARQL para *XQuery*, não há muitos trabalhos que foquem exatamente neste problema. Em geral, a maioria das abordagens se concentra na transformação de dados XML para RDF, e vice-versa, utilizando, para isso, uma combinação de tecnologias da Web Semântica com tecnologias XML. Na linha de pesquisa voltada para *Serviços Web*, por exemplo, o grupo do W3C que trabalha com Anotações Semânticas para WSDL (SAWSDL) [Farrell and Lausen 2007] utiliza XSLT para converter dados XML em RDF. Este processo é chamado de *lifting*, ao passo que a transformação inversa (*lowering*) é realizada através da utilização de SPARQL e XSLT. De maneira semelhante, outros grupos do W3C (GRDDL) [Connolly 2007] utilizam XSLT para extrair dados RDF de documentos XML. Por outro lado, há trabalhos que buscam permitir que dados XML e RDF possam ser consultados em paralelo. Para isto, Groppe [Groppe et al. 2008] propõe incorporar subconsultas SPARQL em XQuery/XSLT, enquanto Droop [Droop et al. 2008] inclui consultas *XPath* em SPARQL.

Como é possível observar, a maioria das abordagens citadas utiliza a linguagem XSLT para transformar dados, ou ainda, recebe como entrada tipos específicos de consultas XQuery ou XPath, combinadas com construtores de SPARQL, visando

consultar documentos XML e RDF em paralelo, o que não se aplica à nossa proposta. Com isso, os dois trabalhos descritos a seguir se enquadram melhor em nosso ambiente.

Em [Bikakis et al. 2009b], os autores, de posse da ontologia local gerada, descrevem um método que traduz uma consulta SPARQL em uma consulta *XQuery* sobre esta ontologia. Para isto, utilizam um conjunto de mapeamentos simples expressos em *XPath*. A grande vantagem deste trabalho é permitir a tradução entre as duas linguagens que são, atualmente, os padrões do W3C. Além disso, o fato de recebermos uma consulta SPARQL como entrada facilita a utilização deste método.

Já Akhtar [Akhtar et al. 2008] apresenta a linguagem *XSPARQL*, a qual é resultado da fusão entre SPARQL e *XQuery*. Com isso, podemos facilmente mapear nossas consultas de *SPARQL* para *XSPARQL*, já que a segunda linguagem engloba a primeira, e executar estas consultas sobre uma fonte de dados XML. Um benefício adicional ao se utilizar *XSPARQL* consiste na possibilidade de obter a resposta da consulta diretamente em RDF. Como desvantagem, temos que tal linguagem não é um padrão do W3C, portanto, ainda não há uma consolidação desta, bem como uma vasta quantidade de documentação e APIs.

2.1.2 Mapeamento entre Ontologias

Mapeamentos são asserções que indicam como dados estruturados de acordo com um esquema podem ser transformados em dados descritos por meio de outro esquema. Tais asserções são essenciais para diversos cenários de mediação, como por exemplo, integração de informações, transformação de dados (*data exchange*) e reescrita de consultas [Yu and Popa 2004] . Em geral, duas tarefas principais conduzem o processo de geração de mapeamentos [Euzenat and Shvaiko 2007, de Bruijn and Polleres 2004]:

(i) *Matching*: consiste no processo de encontrar o conjunto de correspondências ou relacionamentos (alinhamento) entre os elementos de diferentes esquemas, neste caso específico, das ontologias. Para esta tarefa, uma série de abordagens tem sido propostas na literatura, as quais utilizam métodos léxicos (para comparação de strings), estruturais e semânticos, além dos baseados na análise de instâncias e interação com o usuário. Em [Euzenat and Shvaiko 2007] é possível encontrar uma revisão completa sobre os métodos e ferramentas existentes para *matching* de ontologias.

(ii) Especificação do Mapeamento: uma vez obtido o alinhamento entre as ontologias, este deve ser expresso através de uma linguagem ou formalismo adequado, de modo que possa ser utilizado por uma determinada aplicação. Tal aplicação irá

definir o nível de complexidade do mapeamento, que pode variar desde simples associações até expressões lógicas complexas. Em [de Bruijn and Polleres 2004], os seguintes fatores são apontados como alguns dos determinantes para a escolha apropriada do tipo de representação a ser utilizada: (i) expressividade e estrutura das ontologias; (ii) grau de automação do processo de geração dos mapeamentos e (iii) nível de precisão dos mapeamentos. Além disso, em geral, mapeamentos podem ser classificados de acordo com as seguintes características:

- Direcionalidade: um mapeamento pode ser unidirecional (injetivo) ou bidirecional (bijetivo). Um mapeamento unidirecional especifica como os termos da ontologia alvo podem ser descritos utilizando os termos da ontologia fonte, de forma que esta associação não é facilmente reversível. Já os mapeamentos bidirecionais podem ser utilizados em ambos os sentidos.

- Heterogeneidade: mapeamentos homogêneos são aqueles que associam entidades de natureza semelhante, como por exemplo, classe com classe e propriedade com propriedade. Os mapeamentos heterogêneos, por outro lado, mapeiam entidades de natureza distinta, permitindo que uma informação representada como uma classe em uma ontologia possa ser descrita como uma propriedade em outra ontologia [Ghidini and Serafini 2006]. Mais especificamente, podemos definir a heterogeneidade de mapeamentos da seguinte forma: considere duas ontologias O_T e O_S e um mapeamento M_{ST} que relaciona os conceitos de ambas. Seja ainda C_T e C_S o conjunto de classes e P_T e P_S o conjunto de propriedades pertencentes ao vocabulário de O_T e O_S, respectivamente. Um mapeamento homogêneo relaciona elementos de C_T apenas com elementos de C_S, assim como P_T com P_S. Já um mapeamento heterogêneo associa elementos de C_T e P_T com elementos de ambos os conjuntos C_S e P_S, ou vice-versa.

Em relação a estes fatores citados, os mapeamentos que adotamos neste trabalho apresentam as características descritas a seguir, conforme detalhado no Capítulo 3. (i) As ontologias são expressas em um dialeto de OWL e podem possuir diferenças estruturais entre si. (ii) O alinhamento entre as ontologias pode ser obtido através de qualquer operador de *match* existente. Uma vez gerado o alinhamento, este é representado através de um modelo, baseado em sêxtuplas, para especificação de *matching*. Finalmente, a partir deste modelo, é gerado um conjunto de regras de mapeamento. Todo este processo ocorre de maneira semi-automática, necessitando de intervenção do usuário somente para validação e refinamento das sêxtuplas. (iii) Consideramos que os mapeamentos obtidos são precisos, corretos e consistentes. No

que diz respeito à direcionalidade, classificamos tais mapeamentos como unidirecionais. Quanto à heterogeneidade, estes podem ser homogêneos ou heterogêneos.

2.1.3 Reescrita de Consultas

A *reescrita de consultas*, na teoria de banco de dados, é uma abordagem para processamento de consultas, na qual, dada uma consulta e um conjunto de definições de visões, o objetivo consiste em reformular a consulta em uma nova expressão. Esta nova expressão, chamada de *consulta reescrita*, deve referir-se às informações presentes nas visões para fornecer uma resposta ao usuário [Calvanese et al. 2000]. O problema de reescrita de consultas é relevante para diversas aplicações, tais como otimização de consultas, *data warehousing* e integração de informações. Particularmente, nos ambientes de integração de informações, as visões são expressas, ou obtidas, a partir de um conjunto de mapeamentos estabelecidos entre os esquemas, com o intuito de ajustar uma consulta submetida sobre o esquema alvo de acordo com um ou mais esquemas fontes. Na literatura, é comum a utilização da nomenclatura *reformulação de consultas* para caracterizar atividades semelhantes, as quais abrangem, algumas vezes, a própria reescrita.

No que tange às ontologias, estas se relacionam com o processo de reescrita através de duas maneiras principais:

(i) Como artefatos semânticos de auxílio à reformulação das consultas. Neste sentido, alguns trabalhos utilizam as ontologias para atividades de expansão, relaxamento e enriquecimento das consultas, bem como para representação de informações sobre contexto e preferências do usuário.

(ii) Como as próprias fontes de dados ou, ainda, representando estas fontes ou o esquema de mediação. Nestes casos, a consulta é submetida diretamente sobre a ontologia e a reescrita é realizada entre as ontologias, ou seja, é necessário lidar com o problema de *reescrita de consultas entre ontologias*. Para isto, alguns aspectos devem ser considerados, como por exemplo, as características dos construtores ontológicos e a sintaxe e semântica das linguagens para representação e consultas sobre ontologias.

Neste trabalho, focaremos no caso (ii), propondo, implementando e validando uma abordagem para o problema de *reescrita de consultas entre ontologias*, considerando diferenças estruturais entre estas. As seções sebsequentes apresentam as principais características e cenários de aplicação deste problema.

Reescrita de Consultas na arquitetura de Mediadores

A arquitetura de mediadores foi introduzida no trabalho apresentado em [Wiederhold 1992], onde um mediador é descrito como um módulo de software cujo objetivo é lidar com as heterogeneidades de representação em sistemas de informação distribuídos. Esta arquitetura tem sido adotada por diversos sistemas de integração de dados, com intuito de fornecer ao usuário uma interface uniforme para acesso e recuperação de informações em fontes de dados distribuídas, sem que este necessite saber detalhes sobre o armazenamento e a recuperação dos dados. Dessa forma, um sistema baseado na arquitetura de mediador é responsável por reformular, em tempo de execução, uma consulta do usuário submetida sobre o esquema de mediação, também chamado de esquema global, em várias subconsultas sobre as fontes de dados pertencentes ao sistema. Para este fim, são estabelecidos mapeamentos que especificam o relacionamento entre estas fontes de dados e o esquema global. A abordagem adotada para definição dos mapeamentos determina como o processo de reescrita de consultas será desenvolvido no sistema [Lenzerini 2002].

A *abordagem GAV* requer que o esquema global seja expresso em termos das fontes de dados, ou seja, cada elemento *g* do esquema de mediação é associado a uma visão q_S sobre as fontes de dados. Esta abordagem facilita o processo de reescrita, de forma que uma consulta submetida sobre o esquema global é respondida por meio do desdobramento da consulta (*unfolding*), também chamado de expansão das visões [Bilke 2007]. Neste processo, as entidades do mediador presentes na consulta do usuário são substituídas por suas respectivas definições nos mapeamentos, resultando em uma consulta contendo somente termos dos esquemas das fontes. Na *abordagem LAV*, o conteúdo de cada fonte de dados é caracterizado em termos do esquema global, ou seja, cada elemento *s* pertencente a um dos esquemas fontes é associado a uma visão q_G sobre o esquema de mediação. Esta abordagem favorece a manutenção e a extensão do sistema, mas torna o processo de reescrita mais custoso, uma vez que este não pode ser realizado através da estratégia de *unfolding*, como na GAV. Neste caso, o problema consiste em como responder a uma consulta definida sobre o esquema global utilizando somente visões que descrevem as fontes de dados. Por fim, podemos mencionar outras duas abordagens (GLAV [Madhavan and Halevy 2003] e BAV [Mcbrien and Poulovassilis 2007]), as quais combinam GAV e LAV, visando tirar proveito dos benefícios de ambas.

Dadas estas características fundamentais, os sistemas de integração de dados com mediadores evoluíram e passaram a adicionar ontologias em suas arquiteturas, com intuito de lidar com a heterogeneidade semântica. Na literatura, é possível identificar duas arquiteturas gerais para integração de dados baseada em ontologias [Wache et al. 2001]: arquitetura de dois níveis (Figura 2.3(a)) e arquitetura de três níveis (Figura 2.3(b)).

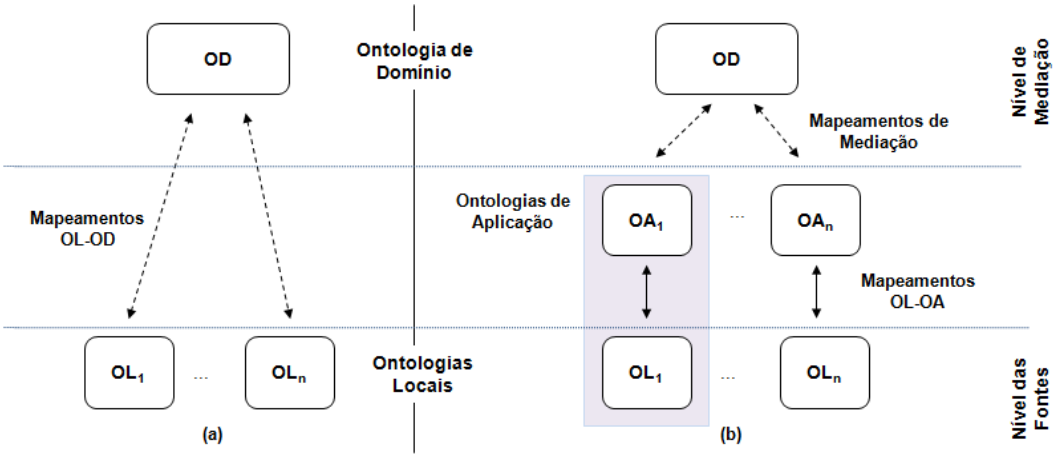


Figura 2.1. Arquitetura de dois (a) e três (b) níveis para integração de dados com ontologias

Em ambas as arquiteturas, há a existência de: (i) uma ontologia de domínio que, além de representar o esquema de mediação, fornece uma representação conceitual de um domínio, como suas respectivas restrições associadas; (ii) fontes de dados que encontram-se representadas através de ontologias locais; (iii) um conjunto de mapeamentos entre as ontologias. O que distingue estas arquiteturas é a introdução, na arquitetura de três níveis, das ontologias de aplicação. Tais ontologias representam um subconjunto da ontologia de domínio e são utilizadas para facilitar as tarefas de recuperação e integração das informações, uma vez que permitem a divisão dos mapeamentos em dois estágios: (i) mapeamentos homogêneos no nível de mediação e (ii) mapeamentos OL-AO, que podem ser mais complexos e heterogêneos. Os primeiros possibilitam que a resposta a uma consulta sobre várias fontes de dados seja efetuada por meio de simples uniões, enquanto os segundos lidam com o problema da reestruturação de informações, porém para uma fonte de dados apenas. Esta divisão favorece o processo de reescrita em ambientes de integração, principalmente quando as ontologias locais apresentam diferenças estruturais significativas.

Para ilustrar esta situação, considere os trechos de três ontologias apresentados abaixo (Figuras 2.4 e 2.5). Seja ainda seguinte notação: *Classe(x)* representa um

conceito cujo nome é *Classe* e *x* corresponde à URI de uma determinada instância de *Classe*; *propriedade(x,y)* representa uma propriedade cujo domínio é uma instância com URI igual a *x* e o range é um valor literal *y* (ou uma URI).

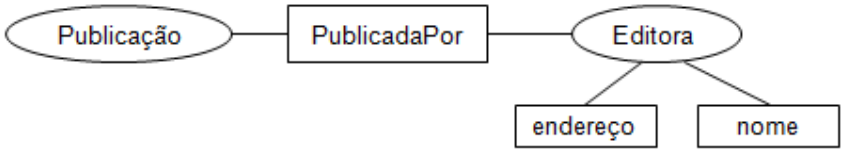


Figura 2.2. Ontologia de domínio O_D

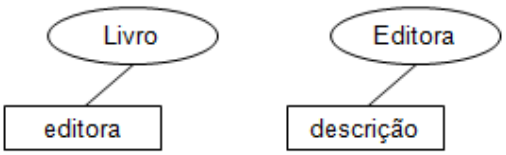


Figura 2.3. Ontologias locais O_{L1} e O_{L2}

Na arquitetura de duas camadas, não seria possível definir o seguinte mapeamento (estilo GAV) no nível de mediação: O_D*: Editora(x)* ← O_{L1}*: editora(x,y)* ∪ O_{L2}*: Editora(x)*. Isto porque tanto na ontologia de domínio O_D quanto na ontologia O_{L2}, *Editora* corresponde a uma classe, enquanto em O_{L1}, refere-se a uma propriedade. Observe que *x* em O_{L1} não é uma instância de *Editora*, mas sim de *Livro*. Dessa forma, se dividirmos a definição do mapeamento da seguinte maneira, podemos estabelecer a associação desejada:

O_D*: Editora(x)* ← O_{A1}*: Editora(x)* ∪ O_{A2}*: Editora(x)*

O_{A1}*: Editora(f(y))* ← O_{L1}*:editora(l,y)*, O_{L1}*:Livro (y)*

O_{A2}*: Editora(x)* ← O_{L2}*: Editora(x)*

Neste ponto, salientamos a idéia por trás do uso das OAs: possibilitar que as fontes de dados apresentem suas informações de acordo com a ontologia de domínio, de forma a facilitar o processo de reescrita para as várias fontes, bem como a unificação dos resultados. Note que a partir do momento em que se estabelecem os mapeamentos com reestruturação, este objetivo é satisfeito, o que indica que a base conceitual da arquitetura de três camadas apresentada em [Sacramento et al. 2010a] reside neste aspecto.

Neste cenário, a abordagem apresentada neste trabalho pode ser aplicada para reescrever consultas entre as ontologias de aplicação e as ontologias locais, conforme destacado na Figura 2.3.

Reescritas de Consultas em Arquiteturas P2P

Sistemas de Gerenciamento de Dados *peer-to-peer* (*PDMS – Peer Data Management System*) surgiram como uma extensão natural dos sistemas de integração de dados baseados em mediadores. Os PDMS são considerados o resultado da união dos benefícios do paradigma P2P, tais como a perda de uma unidade centralizadora, com a semântica rica dos bancos de dados [Pires 2009]. Devido às características intrínsecas da arquitetura P2P, nestes sistemas não há a existência de um único esquema global. Ao invés disso, cada *peer* representa uma fonte de dados e fornece um esquema exportado, o qual corresponde aos dados a serem compartilhados com os demais *peers* do sistema. Além disso, os mapeamentos são estabelecidos entre estes esquemas exportados, com intuito de possibilitar a troca de informações e a reescrita de consultas. Os PDMS tem sido utilizados para troca de dados (*data exchanging*), resposta à consultas (*query answering*) e compartilhamento de informações em muitos domínios de aplicação, como por exemplo, na pesquisa científica e em sistemas educacionais [Green et al. 2007; Ng et al. 2003].

A Figura 2.6 ilustra um exemplo de PDMS. Nele, cinco fontes de dados (*FD*) estão conectadas, através de seus respectivos esquemas de exportação (*EXP*). As setas representam os mapeamentos estabelecidos entre tais esquemas. Neste sistema, cada nó desempenha ambos os papéis de cliente e servidor, significando que se um usuário submeter uma consulta baseada no esquema de um determinado *peer*, o resultado irá ainda conter dados de todos os outros *peers* relevantes para a resposta.

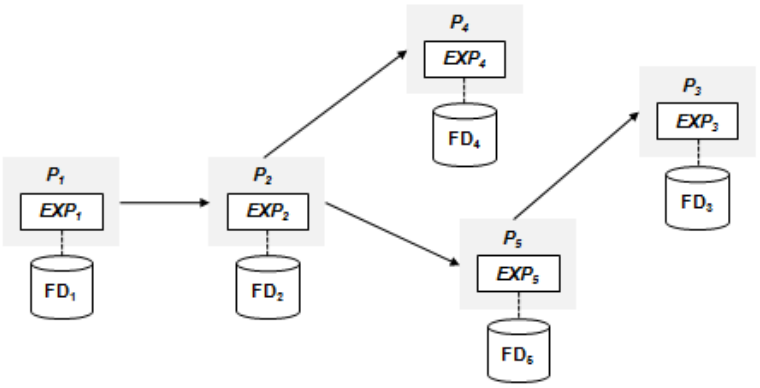


Figura 2.4. Exemplo da arquitetura de um PDMS

Para obter esta resposta, o processamento da consulta, ilustrado na Figura 2.7, é desenvolvido conforme descrito a seguir. Seja *Q* uma consulta submetida sobre o *peer* P_1, de acordo com o esquema exportado EXP_1. A consulta *Q* é, inicialmente, reescrita para uma consulta *Q'* sobre o(s) vizinho(s) imediato(s) de P_1, que neste caso, é o *peer* P_2. O conjunto de mapeamentos M_{1-2} é utilizado durante este processo. Em seguida, *Q'* é reescrita em outras duas consultas *Q''* e *Q'''*, respectivamente, sobre os *peers* P_5 e P_4, considerando os mapeamentos M_{2-5} e M_{2-4}. Finalmente, *Q'''* é reformulada de acordo com o esquema do *peer* P_3, utilizando o mapeamento M_{5-3}. Após as consultas serem executadas em cada um dos *peers* visitados, os resultados são enviados destes para o *peer* onde a consulta inicial foi submetida (P_1). Tal *peer* é responsável por integrar os resultados e apresentar a resposta ao usuário. Como é possível observar, a reescrita da consulta é desenvolvida múltiplas vezes, sempre entre um esquema alvo e um esquema fonte.

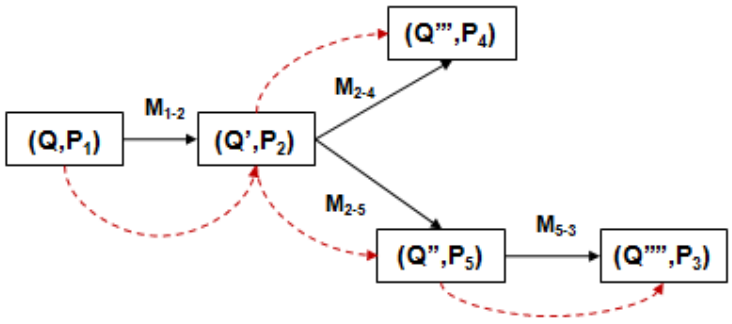


Figura 2.5. Processamento de uma consulta em um PDMS

É válido mencionar ainda que, em muitos casos, para responder a uma consulta submetida pelo usuário, é possível utilizar vários caminhos distintos entre os *peers*. Encontrar um bom caminho, o qual possibilite uma menor perda de informação e um menor custo computacional, é um desafio neste tipo de ambiente [Bilke 2007]. Além disso, por questões de escalabilidade, a distribuição da consulta pode ser restrita a um número limitado de *peers*.

Uma vez que os PDMS também lidam com fontes de dados heterogêneas, ontologias podem ser utilizadas como um modelo de dados comum para descrever os esquemas de exportação. Com isso, é possível representar os *peers* através de uma notação conceitual uniforme, o que facilita a definição dos mapeamentos entre os esquemas e, consequentemente, melhora o processamento da consulta [Xiao and Cruz 2006]. Esta união dos conceitos relativos aos PDMS com as ontologias leva a geração dos Sistemas de Gerenciamento de Dados *peer-to-peer* baseado em Ontologias

(OPDMS), cujo principal objetivo é possibilitar a interoperabilidade semântica entre os *peers*. Nesse sentido, destacamos que a vinculação de ontologias de aplicação aos *peers* pode facilitar a interoperabilidade neste sistema, caso haja uma ontologia de domínio associada ao PDMS.

Por fim, salientamos que além da topologia descrita nesta seção (sistema *peer-to-peer* puro), há ainda outras topologias, dentre as quais, a *super-peer*. Nesta topologia, um *peer* especial, chamado de *super-peer*, atua como um servidor para outros *peers*, podendo desempenhar tarefas complexas, como por exemplo, responder a consultas e integrar dados. No caso específico de PDMS, um *super-peer* pode ser visto como um mediador para um grupo de diversos *peers*, podendo estar ligado a outros *peers* e *super-peers* do sistema.

Reescrita de Consultas em *Linked Data*

Nos últimos anos, um número crescente de fontes de dados no formato RDF tem sido publicadas na *Web*, principalmente devido aos esforços da comunidade envolvida no projeto *Linked Data*[2]. O objetivo desta iniciativa consiste em delimitar um conjunto de melhores práticas para publicar e conectar dados estruturados na *Web*, com o intuito de criar uma "*Web* de dados". Entre tais práticas, estão: (i) o uso de URIs para identificação dos recursos; (ii) a utilização de tecnologias, tais como RDF e SPARQL, para descrição e consulta a estes recursos, respectivamente; (iii) o reaproveitamento de URIs, de forma que seja possível estabelecer ligações entre os dados disponíveis, com a finalidade de navegar através destas ligações.

Se, por um lado, a *Web de dados* ligados traz facilidades para acesso e navegação, o problema de heterogeneidade não é eliminado. Ao contrário, este problema é transferido do nível sintático para o semântico, em dois aspectos principais: (ii) Idealmente, uma única URI deveria identificar sempre o mesmo recurso. Entretanto, ainda não há uma maneira eficaz de controlar a adoção de URIs. Como consequência, quanto mais *datasets* são publicados na *Web*, maior é o problema de sobreposição de URIs, ou seja, diferentes URIs se referem a uma mesma entidade real. (ii) Assim como as URIs, a adoção de ontologias na *Web* Semântica não pode ser facilmente coordenada, uma vez que diferentes aplicações publicam seus dados utilizando diferentes ontologias.

Para resolver o problema (i), alguns serviços de resolução de co-referência tem sido desenvolvidos e disponibilizados na *Web*. Um exemplo deste tipo de serviço é o

[2] http://linkeddata.org/

semeas[3], que recebe como entrada uma URI e retorna um conjunto de URIs que identificam o mesmo recurso em alguns *datasets* do *Linked Data*. No que diz respeito ao problema (ii), se faz necessária a realização de tarefas de mediação entre as ontologias adotadas. Tais tarefas incluem a geração de mapeamentos, a tradução de dados e a reescrita de consultas.

É importante destacar que, dada a natureza do ambiente, não devemos esperar que uma estratégia de reescrita seja aplicada sobre toda a *Web* de dados, por questões de escalabilidade. Isto seria equivalente a tentar construir um ambiente de integração de dados para toda a *Web*. Conforme discutido em [Bizer et al. 2009], a maneira mais eficiente de se consultar a "*Web* de Dados", de uma maneira mais global, seria utilizando serviços de rastreamento (*crawling*) e armazenamento (*caching*), através de motores de busca guiados por URIs.

No entanto, a integração de um determinado conjunto de *datasets*, expressos em RDF, é possível através da reescrita de consultas utilizando a linguagem SPARQL.

2.2 Abordagens para Reescrita de Consultas com Ontologias

Nesta seção, descrevemos, com maiores detalhes, alguns dos principais trabalhos que se relacionam ou se aproximam de nossa abordagem.

2.2.1 *What to Ask to a Peer*

Em [Calvanese et al. 2004], Calvanese e seu grupo discutem que, nos últimos anos, questões relativas à cooperação, integração e coordenação em sistemas de informação tem sido trabalhadas em diversos contextos, incluindo integração de dados, Web Semântica e ambientes *Peer-to-Peer*. Segundo os autores, de uma maneira abstrata, todos estes sistemas são caracterizados por uma arquitetura constituída por vários "nós" autônomos (chamados de sites, fontes ou *peers*), os quais armazenam informações e estão ligados a outros nós por meio de mapeamentos. Nesta arquitetura, dois problemas básicos, comuns a todos estes sistemas, devem ser investigados: (i) como descobrir e expressar os mapeamentos entre os *peers*; (ii) como explorar estes mapeamentos, com o intuito de responder a consultas submetidas sobre um *peer*. No trabalho apresentado, os autores estudam o segundo problema.

[3] http://sameas.org

Para isto, consideram um cenário simplificado, composto apenas por dois *peers*, chamados de *peer local* (P_l) *e peer remoto* (P_r), onde:

- As consultas são submetidas sobre o *peer* local.
- Há um conjunto de mapeamentos relacionando informações do *peer* remoto com informações do *peer* local
- Cada *peer* fornece um serviço de resposta à consulta sobre sua própria base de conhecimento.
- Uma base de conhecimento de um *peer* é definida como uma tupla da forma P = (K,V,M), onde:
  - K é uma base de conhecimento escrita em algum subconjunto da lógica de primeira ordem (LPO), com um alfabeto composto de constantes e um conjunto de relações (funções não são consideradas no trabalho).
  - V é o fragmento exportado de K.
  - M é um conjunto finito de assertivas de mapeamento da forma $q_r \rightarrow q_l$, onde q_r e q_l são duas consultas de mesma aridade, chamadas consulta de remota e local, respectivamente.
- Uma consulta *q* é aceita por um *peer* P se ela é expressa em termos do alfabeto de V

A partir deste cenário, é descrito formalmente o problema "*What-to-Ask*": dado dois *peers* P_l e P_r e uma consulta *q* sobre P_l, encontre uma maneira de responder a esta consulta utilizando apenas o conjunto de mapeamentos entre P_l e P_r, bem como os serviços de resposta à consulta existentes em cada um destes *peers*.

Definido o problema genérico, os autores o especializam para o caso onde a base de conhecimento dos *peers* é expressa através de uma linguagem básica para representação de ontologias. A linguagem utilizada é um subconjunto de LPO que captura os seguintes construtores: constantes (instâncias), predicados unários (classes), predicados binários (propriedades ou relacionamentos), tipagem de relacionamentos (domínio e *range*), participação obrigatória de um indivíduo de uma classe em um relacionamento (restrição de cardinalidade 0 e 1), funcionalidade de relacionamentos (propriedades funcionais) e subsunção entre classes.

Os mapeamentos são representados da seguinte forma: $q_r \rightarrow \{x \mid C(x)\}$ ou $q_r' \rightarrow \{x_1, x_2 \mid R(x_1,x_2)\}$, onde q_r e q_r' são consultas sobre o fragmento exportado do *peer* remoto (P_r,) e *C* e *R* são conceitos e relacionamentos, respectivamente, do *peer* local (P_l). Na terminologia de integração de dados, estes mapeamentos podem ser

considerados GAV, de forma que P_l corresponderia ao esquema global e P_r, ao conjunto de fontes de dados. Além disso, é assumido que para cada conceito C ou relacionamento R do *peer* local há, no máximo, uma assertiva de mapeamento que utiliza C (ou R). No que se refere às consultas aceitas, estas devem ser consultas conjuntivas constituídas por relações, variáveis e constantes que se refiram ao vocabulário do *peer* correspondente.

Visando resolver o problema *What to Ask* para ontologias, é proposto um algoritmo chamado de *computeWTA*. Em suma, este algoritmo, primeiramente, reformula a consulta q do cliente em um conjunto Q de consultas conjuntivas, expressas sobre K_l, utilizando informações contidas na base de conhecimento. Mais especificamente, as restrições de subsunção, tipagem e funcionalidade são utilizadas para expandir o conjunto Q. Em seguida, de acordo com o mapeamento M_l, o algoritmo reescreve as consultas contidas em Q em um conjunto de consultas aceitas pelo *peer* remoto P_r. Para este fim, para cada consulta $q \in Q$, ele realiza o casamento, entre os átomos de q com os átomos de M_l, gerando uma consulta expressa em termos de K_r.

Finalmente, os autores demonstram formalmente que, utilizando a linguagem de ontologias descrita, aliada às condições apresentadas, é possível computar soluções para o problema *What to Ask*, ou seja, que o algoritmo *computeWTA* é correto e sempre termina. Por outro lado, eles ainda provam que nem sempre haverá solução para o problema, caso seja adicionada a subsunção entre relacionamentos (propriedades). Isto porque o uso deste construtor pode levar à geração de um número infinito de consultas reescritas sobre P_r.

2.2.2 *SomeRDF*

Em [Adjiman et al. 2007], os autores argumentam que a *Web* Semântica pode ser vista como um grande PDMS semântico, onde as ontologias que representam os dados, juntamente com os mapeamentos entre estas, constituem uma enorme rede P2P. Com isso, eles apresentam um PDMS desenvolvido para *Web* Semântica. Tal PDMS, nomeado de *SomeRDF*, é construído no topo da infra-estrutura *SomeWhere* [Adjiman et al. 2006], utilizando um modelo de dados baseado em RDF. *SomeWhere* é um PDMS onde cada *peer* é visto como um conjunto de cláusulas proposicionais. Dessa forma, o problema de responder a uma consulta (o que inclui a reescrita desta consulta) submetida sobre um *peer* do sistema é reduzido ao problema de encontrar as conseqüências (*consequence finding*) de uma certa fórmula, na teoria da lógica

proposicional, aplicado a um cenário distribuído. Neste caso, esta fórmula pode ser vista como a consulta expressa utilizando o vocabulário de um dado *peer*.

No *SomeRDF*, tanto as ontologias quanto os mapeamentos são expressos através de um fragmento de RDFS, chamado de *core-RDF*, o qual permite a definição de (sub)classes, (sub)propriedades, bem como a descrição do domínio e do *range* das propriedades. O sistema descrito é constituído por um conjunto de: (i) ontologias, chamadas de ontologias dos *peers*; (ii) descrições dos conteúdos armazenados (*storage descriptions*) nos *peers*, isto é, as instâncias das ontologias; (iii) mapeamentos entre os *peers*. Um mapeamento é uma sentença de igualdade ou inclusão entre classes ou propriedades de dois *peers* distintos ou, ainda, uma sentença que define o domínio ou *range* de uma propriedade de um *peer* como sendo uma classe de outro *peer* (veja Tabela 2.5). Todos os construtores do sistema são representados na semântica da lógica de primeira ordem (LPO) sem considerar a presença de funções. A Tabela 2.5 ilustra a representação das ontologias, instâncias e mapeamentos.

As consultas, também representadas em lógica de primeira ordem, são consultas conjuntivas que podem envolver o vocabulário de vários *peers*. Entretanto, a consulta submetida pelo usuário deve ser formulada em termos de um único *peer*.

O processo de reescrita da consulta entre as ontologias é realizado através de um algoritmo, chamado de $DeCA^{RDFS}$, através do qual os autores convertem o conhecimento expresso em LPO para a teoria proposicional e lidam com o problema de reescrita de maneira semelhante ao *SomeWhere*. Para este fim, o $DeCA^{RDFS}$ age como uma interface entre o usuário e o algoritmo *DeCA* (*Decentralized Consequence finding Algorithm*), utilizado no *SomeWhere*.

Tabela 2.5. Representação das ontologias e dos mapeamentos no *SomeRDF*

Elementos da Ontologia		
Construtor	***Notação em DL***	***Notação em LPO***
Inclusão de classes	$C_1 \sqsubseteq C_2$	$\forall x(C_1(x) \Rightarrow C_2(x))$
Inclusão de propriedades	$P_1 \sqsubseteq P_2$	$\forall x \forall y(P_1(x,y) \Rightarrow P_2(x,y))$
Tipo do domínio	$\exists P \sqsubseteq C$	$\forall x \forall y(P(x,y) \Rightarrow C(x))$
Tipo do r*ange*	$\exists P^- \sqsubseteq C$	$\forall x \forall y(P(x,y) \Rightarrow C(y))$
Instância de uma classe	C(a)	
Instância de uma propriedade	P(a, b)	
Mapeamento entre as Ontologias		
Mapeamento entre O1 e O2	***Notação em DL***	***Notação em LPO***
Classe	$P_1:C_1 \sqsubseteq P_2:C_2$	$\forall x(C1(x) \Rightarrow C2(x))$
Propriedade	$P_1:P_1 \sqsubseteq P_2:P_2$	$\forall x \forall y(P1(x,y) \Rightarrow P2(x,y))$
Domínio	$\exists P_1:P \sqsubseteq P_2:C$	$\forall x \forall y(C1(x) \Rightarrow C2(x))$
Range	$\exists P_1: P^- \sqsubseteq P_2:C$	$\forall x \forall y(C1(x) \Rightarrow C2(x))$

A estratégia do $DeCA^{RDFS}$ é reescrever, de maneira independente, cada átomo da consulta do usuário utilizando o *DeCA* e, ao final, combinar todas as reescritas, com o intuito de gerar as reescritas conjuntivas da consulta do usuário. Na literatura [Goasdoué and Rousset 2004], é demonstrado que se o número máximo de reescritas conjuntivas de uma consulta é finito, então a resposta desta consulta pode ser obtida a partir da união dos resultados de suas reescritas. Diante disso, o algoritmo $DeCA^{RDFS}$ garante que o número máximo de reescritas conjuntivas da consulta do usuário, no que se refere às informações dos esquemas e dos mapeamentos, é gerado. Além disso, os autores ainda demonstram que o algoritmo é correto, completo e sempre termina.

2.2.3 *SemRef* (*Semantic Reformulation*)

SemRef (*Semantic Reformulation*) [Fernandes 2009] é uma abordagem para reformulação de consultas, em ambientes dinâmicos e distribuídos, entre uma ontologia fonte e uma ontologia alvo. No trabalho, a autora argumenta que um problema comum neste tipo de ambiente diz respeito ao fato dos conceitos existentes em um *peer* fonte nem sempre terem uma correspondência exata no *peer* alvo. Como resultado, há a geração de um conjunto vazio de reformulações de algumas consultas e, possivelmente, a ausência de respostas ao usuário. Diante disso, dependo das preferências deste usuário, pode ser mais vantajoso produzir uma reformulação adaptada/enriquecida, que devolva respostas próximas ao resultado desejado, ao invés de não retornar resposta alguma.

Para produzir estas consultas enriquecidas, além da equivalência ($\equiv$), as seguintes correspondências semânticas são utilizadas entre os conceitos das ontologias fonte e alvo: especialização ($\sqsubset$), generalização ($\sqsupset$), proximidade ($\approx$), parte-de ($\rhd$), conjunto-de ($\lhd$) e disjunção ($\perp$). Tais correspondências são obtidas a partir do *matching* entre as ontologias que representam os *peers* (que devem pertencer ao mesmo domínio de discurso) e uma ontologia de domínio, a qual deve conter a semântica do domínio da aplicação. Com relação a estas correspondências, um diferencial refere-se à utilização das relações de proximidade e disjunção. A primeira, quando o usuário habilita o uso de consultas aproximadas, permite que conceitos relacionados aos originalmente expressos na consulta possam ser utilizados durante o processo de expansão desta. Já a segunda, provê uma solução para a negação de um conceito, através da substituição deste por todos os conceitos disjuntos a ele.

Além disso, são apontadas como características fundamentais do método proposto, a consideração do contexto do usuário, da consulta e do ambiente. O contexto do usuário é obtido por meio de um conjunto de variáveis de enriquecimento, nas quais usuário pode definir os níveis de aproximação da consulta, caso este deseje a geração de consultas aproximadas, além das exatas. O contexto da consulta é identificado a partir da semântica e do modo de reformulação desta. O contexto do ambiente é adquirido com base na disponibilidade dos *peers*, considerando tanto o *peer* onde a consulta foi submetida quanto os vizinhos deste *peer*.

O algoritmo *SemRef* recebe como entrada uma consulta Q, submetida sobre uma ontologia O_1, as ontologia O_1 e O_2, um conjunto de correspondências semânticas entre O_1 e O_2 ($Co[O_1,O_2]$) e o contexto informado pelo usuário (variáveis de enriquecimento e modo de reformulação da consulta). Com saída, o algoritmo produz uma ou duas consultas reformuladas (exata e/ou enriquecida). As ontologias utilizadas são representadas no subconjunto da Lógica Descritiva conhecido como $\mathcal{ALC}$-DL [Baader et al. 2007]. Uma consulta submetida sobre uma ontologia O_i é uma expressão da forma: $Q = Q_1 \sqcup Q_2 \sqcup ... \sqcup Q_M$, onde $Q_i = C_1 \sqcap C_2 \sqcap ... \sqcap C_N$ e cada conceito $C_i \in \{C_j, \sim C_j, \forall R.C_j, \exists R.C_j\}$. Isto significa que as consultas são expressas na forma normal disjuntiva, isto é, correspondem à união (disjunção) de consultas conjuntivas. Já os mapeamentos são homogêneos e representados em DDL (*Distrubuted Description Logic*) [Ghidini and Serafini 2006].

A Tabela 2.6 ilustra um exemplo com um conjunto de correspondências Co_{12} da ontologia O_1 para a ontologia O_2, bem como uma consulta Q submetida sobre O_1 e as respectivas consultas reescritas (exata e enriquecida) sobre O_2.

Em suma, para cada conceito C_j existente na consulta, o algoritmo busca as correspondências do tipo C_j ■ C' (■ representa a operação), adicionando o conceito C' na consulta reformulada. Este processo é desempenhado respeitando as preferências do usuário. Por exemplo, se este não habilitou as variáveis de enriquecimento, então a única correspondência utilizada é a de equivalência, o que leva a geração somente de consultas exatas. Caso contrário, as demais correspondências são também utilizadas.

O método proposto é validade através da construção de um protótipo (que recebe consultas em DL e SPARQL), o qual foi utilizado para realização de experimentos. Além disso, é apresentada uma demonstração formal simples acerca da corretude e da completude do algoritmo *SemRef*.

Tabela 2.6. Exemplo de um conjunto de correspondências e consultas utilizadas na abordagem *SemRef*

Conjunto de Correspondência entre O_1 e O_2
O_1:FullProfessor $\equiv$ O_2:FullProfessor
O_1:FullProfessor $\sqsubseteq$ O_2:Professor
O_1:FullProfessor $\approx$ O_2:VisitingProfessor
O_1:FullProfessor $\perp$ O_2:AssociateProfessor
O_1:FullProfessor $\rhd$ O_2:Course
O_1:FullProfessor $\rhd$ O_2:ResearchProject
Consultas
Q = FullProfessor
Q_{EXATA} = [FullProfessor]
$Q_{ENRIQUECIDA}$ = [VisitingProfessor ⊔ Professor ⊔ Course ⊔ ResearchProject]

2.2.4 *Linking Data to Ontologies*

No trabalho apresentado em [Poggi et al. 2008; Calvanese et al. 2009], os autores conduzem um estudo teórico acerca do problema de OBDA, levantando questões sobre a complexidade computacional de Lógicas Descritivas, Modelagem Conceitual, entre outros. Além disso, é apresentada uma abordagem para resolver este problema, considerando: (i) que as ontologias estão expressas na linguagem *DL-Lite*$_A$ (um subconjunto de DL proposto pelos autores), a qual contém os principais construtores para modelagem conceitual (classes, relacionamentos, atributos, domínio, *range*, subsunção e funcionalidade); (ii) que os dados estão armazenados em SGBDs relacionais. Com isso, para lidar com a questão de acessar dados relacionais através de uma ontologia, é proposta uma solução para o problema de incompatibilidade de impedância (*mismatch impedance*), que surge devido à necessidade de se extrair objetos, que são as instâncias da ontologia, a partir de valores presentes na fonte de dados.

A resposta a uma consulta submetida sobre a ontologia é obtida através da utilização um conjunto de mapeamentos, onde cada conceito ou relacionamento da ontologia é associado a uma consulta em SQL. Para a obtenção desta resposta, a consulta é reescrita e executada sobre a base de dados, através dos seguintes passos: (i) **reformulação**, (ii) **desdobramento (*unfolding*)** e (iii) **execução**. Estas atividades estão ilustradas na Figura 2.8 e ocorrem conforme descrito a seguir. Inicialmente, uma consulta *Q*, submetida sobre ontologia, é expandida de acordo com as restrições

presentes na própria ontologia, como por exemplo, restrições hierárquicas (subclasse, superclasse), de domínio, *range* e funcionalidade. Portanto, durante esta etapa, a consulta é reformulada levando em consideração somente os componentes do *TBox*. Como resultado, é gerada uma nova consulta *Q'*, a qual representa a união das consultas conjuntivas obtidas durante a **reformulação**. No entanto, *Q'* não é diretamente utilizada. Tal consulta serve como entrada para a próxima etapa, chamada de ***unfolding***, onde *Q'* é traduzida para uma nova consulta *Q''*, em termos do esquema da fonte de dados, considerando o conjunto de mapeamentos entre a ontologia e esta fonte. Finalmente, na etapa de **execução**, *Q''* é submetida sobre a base de dados relacional e seus resultados são retornados ao usuário.

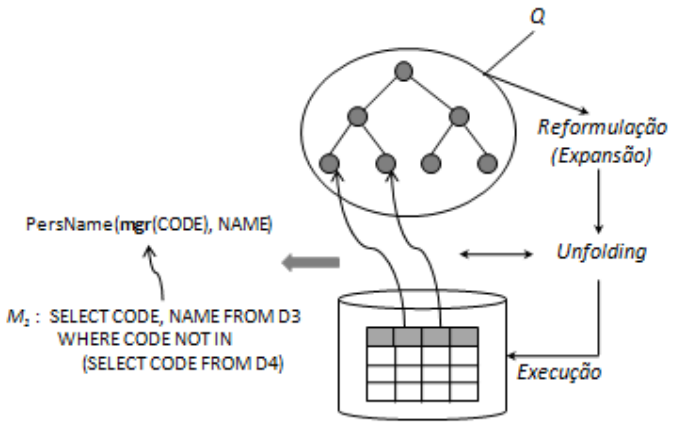


Figura 2.6. Abordagem para acesso a dados relacionais a partir de ontologias

Para implementação das soluções propostas, os autores desenvolveram o sistema *QuOnto*, que permite ao usuário especificar, manualmente, um conjunto de mapeamentos. Para tanto, o usuário deve escrever consultas SQL sobre a fonte de dados e associá-las com as entidades da ontologia. Além disso, o sistema ainda provê outras funcionalidades, dentre as quais: checagem da consistência de uma ontologia e recebimento de consultas conjuntivas que podem ser respondidas acessando uma fonte de dados externa, conforme descrito.

2.2.5 Reescrita de consultas SPARQL para Integração de *Linked Data*

Correndo [Correndo et. al 2010] apresentam uma abordagem, parcialmente voltada para o contexto dos *datasets* do projeto *Linked Data*, que visa possibilitar a integração de dados RDF através da reescrita de consultas em SPARQL. O algoritmo proposto para a reescrita recebe como entrada uma consulta em SPARQL, uma ontologia *fonte* (*dataset*), que é a ontologia utilizada para formular a consulta, uma ontologia *alvo*

(*dataset*) e um conjunto de alinhamentos entre estas duas ontologias. Além disso, os autores consideram que um *dataset* pode ter mais de uma ontologia vinculada a ele.

Os alinhamentos utilizados são representados por meio de uma quádrupla (chamada de OA – *Ontology Alignments*) da forma OA = (SO, TO, TD, EA), onde: (i) SO é o conjunto de URIs da ontologia fonte; (ii) TO é conjunto de URIs da ontologia alvo; (iii) TD é o conjunto de URIs do *dataset* alvo; (iv) EA é o conjunto de Entidades de Alinhamento de um OA, ou seja, é o alinhamento entre os conceitos presentes nas duas ontologias (fonte e alvo). A Figura 2.9 ilustra um exemplo de uma quádrupla (OA). Em (a), são exibidos os três primeiros conjuntos, enquanto em (b), é possível visualizar uma parte do conjunto EA (o alinhamento da propriedade *akt:has_autor*).

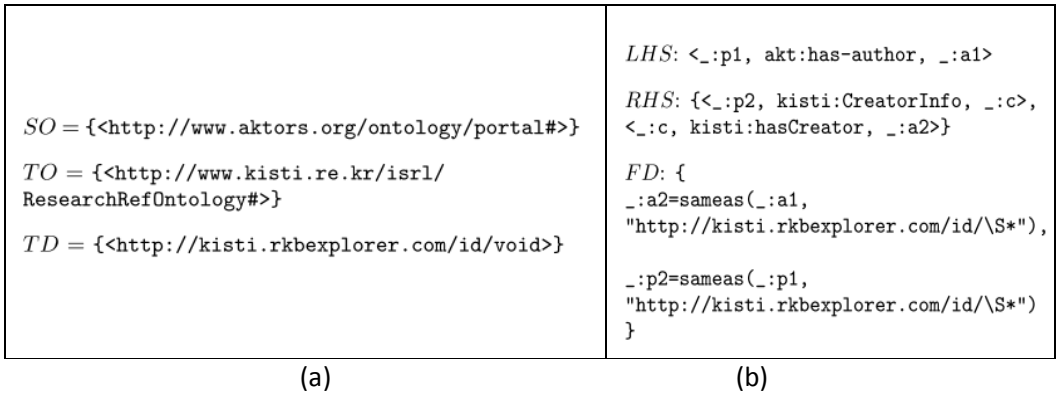


Figura 2.7. Exemplo dos Alinhamentos utilizados por Correndo et. al.

Uma entidade de alinhamento é definida como uma tripla EA = (LHS, RHS e FD), onde: (i) LHS (*Left-Hand Side*) é uma tripla RDF que não contém símbolos funcionais; (ii) RHS (*Right-Hand Side*) é uma conjunção de triplas, também sem símbolos funcionais; (iii) FD (*Functional Dependencies*) é um conjunto de atribuições, para termos presentes na consulta, utilizando funções. No trabalho apresentado, estas dependências funcionais são utilizadas para transformação de URIs, visando lidar com o problema de co-referência de entidade, no contexto da reescrita. Em resumo, isto significa que se, na consulta, houver uma URI representando uma instância, esta será transformada na URI correspondente a tal instância, no *dataset* alvo. Por exemplo, imagine que na consulta haja a seguinte tripla: `?paper akt:has-author id:person-02686`. A URI `id:person-02686` é uma constante que identifica um determinado autor no *dataset* da universidade de *Southampton*. Se a consulta que contém esta tripla necessita ser reescrita para uma consulta sobre a fonte de dados

ReSIST[4], que contém informações sobre publicações acadêmicas, é preciso substituir a URI `id:person-02686` pela identificação deste autor no repositório *ReSIST*. Para esta conversão, os autores utilizam as dependências funcionais (veja um exemplo na Figura 2.9 (b)), as quais fazem uma chamada ao serviço ***sameas,*** que retorna todas as URIs equivalentes, para alguns *datasets* existentes no *Linked Data*, à URI passada como entrada. De posse da nova URI, é possível substituí-la na consulta reescrita. Observe que os autores não tratam o problema de co-referência de entidade para realização da fusão dos dados obtidos como resposta, mas somente para a substituição de URIs relativas a instâncias presentes na consulta.

Durante o processo de reescrita da consulta SPARQL, o padrão de grafo é extraído, de forma que os demais construtores não são considerados (nem mesmo o filtro). Em seguida, cada uma das triplas existentes no padrão é casada com o LHS das entidades de alinhamento (EA). Para cada casamento realizado, a tripla é substituída pelo RHS correspondente ao LHS com o qual a tripla casou. Tudo isso, considerando a ligação das variáveis existentes na consulta e a substituição das entidades presentes nas dependências funcionais descritas anteriormente. Ao final, é gerado um novo padrão de grafo a ser submetido sobre o *dataset* alvo.

Para validação da abordagem, foi desenvolvido um sistema simples, no qual o usuário insere uma consulta SPARQL e seleciona o *dataset* alvo. A partir de então, é gerada uma consulta reescrita, que pode ser executada sobre o SPARQL *endpoint* pertencente a tal *dataset*. Embora os autores descrevam o formato dos alinhamentos da entrada, eles não indicam como o sistema pode ser utilizado ou alimentado com estes alinhamentos, bem como não exibem os resultados das consultas testadas. Eles apenas provêem uma base com alguns alinhamentos entre os *datasets* ECS e *DBPedia*[5] e entre o AKT e o *KISTI*[6].

2.2.6 Reescrita de consultas SPARQL para mediação de consultas entre ontologias mapeadas

Assim como Correndo et. al (trabalho descrito anteriormente), Markis e seu grupo [Markis et. al 2010] propõem uma abordagem para reescrita de consultas SPARQL, através da utilização de um conjunto de mapeamentos entre as ontologias. Entretanto,

[4] http://www.resist-noe.org/

[5] http://dbpedia.org

[6] http://kisti.rkbexplorer.com/

tal trabalho não foca em nenhuma questão específica relacionada aos *datasets* do *Linked Data*. A ideia é que a abordagem possa ser utilizada por um sistema baseado em mediadores para integração de dados RDF.

Para descrever a estratégia adotada, os autores consideram que a ontologia *fonte* é a ontologia sobre a qual a consulta é formulada, de maneira que tal consulta deve ser reescrita em termos da ontologia *alvo*. Entre estas ontologias, há um conjunto de mapeamentos (entre entidades das ontologias fonte e alvo, respectivamente) que podem ser de três tipos: (i) classe e expressão com classe; (ii) propriedade de tipo de dado e expressão com propriedades de tipo de dado; (iii) propriedade de objeto e expressão com propriedades de objeto. Portanto, temos que todos os mapeamentos são homogêneos. Além disso, a relação estabelecida entre as entidades mapeadas pode ser de equivalência (≡) ou subsunção(⊑/⊒).

De posse destes mapeamentos, o processo de reescrita é baseado na reformulação do padrão de grafo da consulta. Nesta abordagem, os operadores presentes neste padrão de grafo (AND, UNION, OPTIONAL, FILTER) permanecem inalterados ao longo do processo, assim como todos os construtores presentes na expressão de FILTRO. Dessa forma, a parte mais importante da estratégia consiste na reescrita dos padrões de triplas que formam o padrão de grafo. Os autores dividem estes padrões de triplas em dois grupos, chamados de *Padrões de Triplas de Dados* e *Padrões de Triplas de Esquema*, respectivamente. O primeiro grupo se refere aos padrões que buscam instâncias na ontologia, como por exemplo, o padrão `?x foaf:autor ?y`. Neste caso, a entidade da ontologia (`foaf:autor`) está presente no predicado, de forma que a resposta irá retornar instâncias da ontologia FOAF. Já o segundo grupo de padrões de triplas corresponde aos que procuram informações sobre o esquema da ontologia, como por exemplo, o padrão `?x rdf:subclassOf foaf:Pessoa`. Observe que, se submetido a um *dataset*, este padrão não retornará dados, mas sim todas as subclasses `rdf:subclassOf` da classe *Pessoa*. Além disso, a entidade da ontologia (`foaf:Pessoa`) se encontra presente no objeto e não no predicado.

Uma vez que um padrão de tripla consiste de três partes (sujeito, predicado e objeto), o procedimento de reescrita é dividido em três etapas, na seguinte ordem, de acordo com o local onde a entidade do padrão de tripla aparece: entidade no predicado, entidade no sujeito e entidade no objeto. Em cada uma destas situações, esta entidade é

buscada nos mapeamentos e substituída pelo(s) conceito(s) corresponde(s) na ontologia alvo. Ao final do processo, a consulta está expressa em termos desta ontologia alvo.

A validação da abordagem é feita de forma teórica, sendo demonstrado que a consulta reescrita preserva a semântica da consulta original. No que se refere à implementação, os autores não entram em detalhes a este respeito, não descrevem o sistema, nem realizam um estudo de caso ou experimentos.

2.3 Análise Comparativa dos Trabalhos Relacionados

Antes de nos aprofundarmos na análise comparativa entre as abordagens que mais se aproximam da nossa, iremos discorrer sobre alguns trabalhos que se concentram em linhas de pesquisa semelhantes.

Necib [Necib 2007] utiliza ontologias para reformular consultas SQL submetidas sobre uma base de dados relacional, visando fornecer resultados mais próximos à intenção do usuário. Para isso, uma consulta SQL recebida como entrada é transformada, por meio da adição (expansão) ou remoção de termos, em outra consulta SQL, que é submetida sobre a fonte de dados. Este processo de reformulação é guiado por um conjunto de mapeamentos pré-estabelecidos entre o banco de dados relacional e uma ontologia de domínio. Há ainda trabalhos que utilizam ontologias para prover a aproximação e/ou relaxamento de consultas [Stuckenschmidt et. al 2005; Fernandes 2009], expressas em DL, entre esquemas distintos.

No que se refere aos trabalhos mais voltados para a comunidade da *Web* Semântica, a reescrita de consultas SPARQL tem sido desenvolvida com vários objetivos, dentre os quais, a otimização, a decomposição [Quilitz and Leser 2008] e a tradução de consultas, bem como a implementação de inferências com Lógica Descritiva. A otimização de consultas SPARQL foca em técnicas de reescrita [Schmidt et. al 2010; Groppe et. al 2009; Perez et. al 2009; Stocker e. al 2008; Bernstein et. al 2007] que minimizam a complexidade de avaliação da consulta sobre uma base de dados. Por outro lado, no campo de inferências com Lógica Descritiva, Jing et. al [Jing et. al 2009] propõem reescrever uma consulta SPARQL com o intuito de implementar um mecanismo de inferência sobre esta consulta. Isto porque SPARQL é uma linguagem que aplica o *casamento de padrões de grafos*, não possuindo, originalmente, nenhum mecanismo de inferência sobre seus construtores. No que tange à tradução de

consultas, alguns trabalhos executam a tradução de SPARQL para SQL e *XQuery*, conforme já discutido na Seção 2.2.1.

É importante destacar que todas estas abordagens citadas podem ser vistas como complementares à nossa, uma vez que possuem foco ou cenário de aplicação distintos. Dada esta visão geral, podemos nos concentrar na comparação entre os trabalhos descritos na seção anterior, os quais são mais próximos de nossa estratégia.

Calvanese e seu grupo desenvolvem, em [Calvanese et al. 2004], um trabalho totalmente teórico, onde são apresentados resultados iniciais básicos importantes. Como este trabalho se baseia em um cenário abstrato, é normal que outras abordagens mais pontuais [Poggi et al. 2008; Calvanese et al. 2009] reaproveitem estes resultados, apresentando também outros resultados teóricos relevantes.

A estratégia de reescrita do *SomeRDF*, por exemplo, possui algumas semelhanças com o trabalho de Calvanese e seu grupo. A diferença fica por conta das seguintes características: (i) os autores transformam a consulta expressa em LPO para a teoria da lógica proposicional, argumentando que sem a utilização de variáveis, é possível obter um melhor desempenho, já que ambos as abordagens não lidam com funções; (ii) o trabalho é voltado para a *Web* Semântica e resultados experimentais são apresentados.

Já Fernandes [Fernandes 2009], por outro lado, busca desenvolver uma reformulação mais semântica, adicionando novos operadores nas correspondências e utilizando informações de contexto. A idéia é permitir que os usuários obtenham respostas aproximadas, além das exatas. Entretanto, embora adicione esta semântica, a autora lida apenas com consultas envolvendo classes. Tais consultas podem ser expressas em DL ou em SPARQL, através da utilização de *templates*, os quais mapeiam os elementos de DL para SPARQL. Com isso, a consulta não pode ser formulada de maneira livre, pois SPARQL possui mais possibilidades de construção que DL.

É válido salientar ainda que a utilização de contexto e operadores de correspondências adicionais (além da igualdade e subsunção), não consiste no foco de nosso trabalho, pois não buscamos gerar consultas aproximadas. Portanto, neste ponto, o trabalho de Fernandes pode ser visto como complementar ao nosso, possuindo um direcionamento distinto.

Em [Poggi et al. 2008; Calvanese et al. 2009], o trabalho apresenta muitos resultados teóricos aplicados à lógica descritiva *DL-Lite*$_A$. Os autores também lidam com o problema de OBDA, considerando uma base de dados relacional. Como já

mencionado, na abordagem aqui apresentada, nós estendemos e adaptamos esta idéia. Assim, não tratamos o problema diretamente para fontes de dados relacionais, mas sim para uma representação destas fontes através de uma ontologia local, gerada automaticamente, com seus respectivos mapeamentos locais. Esta atividade é delegada para trabalhos existentes que tratem deste problema de maneira automática. Além disso, com os dois esquemas expressos como ontologias, é possível realizar as atividades de *matching* e mapeamento de maneira semi-automática. No trabalho de Poogi e Calvanese, os mapeamentos devem ser estabelecidos manualmente entre a ontologia e um esquema relacional. Um outro ponto distinto diz respeito à fase de reformulação da consulta. Tanto em [Calvanese et al. 2004] quanto em [Calvanese et al. 2009], os componentes terminológicos (*TBox*) da ontologia são utilizados para expandir a consulta, antes que os mapeamentos sejam utilizados. Já em nosso trabalho, a fase de expansão (considerando o relacionamento de subsunção) ocorre durante a geração dos mapeamentos. Optamos por esta estratégia pelos seguintes motivos: (i) não é preciso realizar a expansão cada vez que uma consulta é submetida sobre a ontologia; (ii) em alguns casos, nem toda a hierarquia necessita ser expandida, pois, conforme será apresentada no Capítulo 3, nem sempre haverá correspondência para todas as subclasses. Além disso, nas abordagens apresentadas nos trabalhos de Calvanese, a expansão considera outras restrições, além da subsunção. Como exemplo, podemos citar as restrições de domínio. Como ilustração, imagine que a seguinte consulta seja submetida sobre a ontologia: Q = *Estudante(x)*. O sistema deve retornar todas as URIs correspondentes a estudantes. Suponha ainda que *nome* seja uma propriedade de tipo de dado cujo domínio é a classe ESTUDANTE. Para recuperar todos os estudantes, devemos adicionar na consulta original o construtor *nome(x,y)*, de forma que Q' = *Estudante(x)* ∪ *nome(x,y)*. Note que esta adição é importante porque, em ontologias, pode existir uma instância *nome("http://M523","Fernanda")*, ainda que a instância *Estudante*("*http://M523"*) não esteja explicitamente declarada. Para lidar com esta questão, consideramos, a despeito da reescrita, que cada ontologia possui um serviço de resposta à consulta sobre seu próprio vocabulário. Caso seja de interesse da aplicação, este serviço pode ser associado a um motor de inferência (*Racer*[7] ou *Pellet*[8], por exemplo). Esta associação já é oferecida por várias APIs que implementam a linguagem SPARQL. Agimos dessa maneira porque, se ontologia fonte não possuir instâncias, isto

[7] http://www.sts.tu-harburg.de/~r.f.moeller/racer/index.html

[8] http://clarkparsia.com/pellet/

é, representar apenas um reflexo de um esquema relacional ou XML, não há necessidade de proceder com esta expansão (exceto a subsunção). Isto porque os atributos estarão sempre associados aos elementos representados pelas classes (no modelo relacional, as tabelas, por exemplo).

No que tange aos trabalhos comparados até aqui, observamos que nenhum deles recebe uma consulta SPARQL livre na entrada, embora alguns possibilitem o mapeamento de DL para SPARQL. Já as abordagens seguintes ([Correndo et. al 2010] e [Markis et. al 2010]) lidam diretamente com SPARQL. No entanto, Correndo et. al não tratam expressões de filtro, além de utilizarem apenas mapeamentos homogêneos, assim como Markis e sua equipe. Além disso, Markis não apresenta a implementação da ideia proposta, enquanto Correndo não discute aspectos teóricos, mostrando apenas um estudo de caso.

De maneira geral, além das diferenças já citadas, podemos apontar as seguintes características como diferenciais de nossa abordagem:

(i) Focamos no problema de reescrita de consultas entre ontologias que apresentam estruturas distintas. Dessa forma, fazemos uso de mapeamentos heterogêneos expressos em uma extensão de *Datalog* [Hull and Yoshikawa 1990]. Por outro lado, mostramos como realizar a reescrita utilizando tais mapeamentos, que podem ser obtidos de maneira semi-automática, ao contrário da maioria dos trabalhos citados (exceto [Fernandes 2009]).

(ii) Nossos mapeamentos heterogêneos lidam com restrições, as quais são caracterizadas no Capítulo 3, em ambos os esquemas fonte e alvo. Além disso, consideramos a existência de predicados de comparação (>, <, =, etc.) tanto nos mapeamentos quanto nas consultas, sendo estes predicados analisados durante o processo de reescrita. Nenhum dos trabalhos analisados trata estes aspectos.

(iii) A maioria dos trabalhos, exceto [Calvanese 2009], não considera a existência de funções nos mapeamentos. Por outro lado, para Calvanese, estas funções são utilizadas apenas para lidar com as diferenças existentes entre os dados de um SGBD relacional e as instâncias das ontologias. Nós utilizamos as funções para tratar algumas diferenças estruturais.

(iv) Recebemos como entrada uma consulta na linguagem SPARQL e realizamos a reescrita combinando noções de programação em lógica com a semântica dos construtores desta linguagem. Durante o nosso processo, ainda possibilitamos que partes da consulta possam ser descartadas, caso seja constatado que tais partes

retornariam vazio. Por fim, possibilitamos que os resultados sejam apresentados conforme a ontologia alvo.

É importante destacar que o método proposto foi implementado e validado, utilizando o protótipo desenvolvido. A Tabela 2.7 apresenta um resumo das principais características dos trabalhos relacionados, em comparação à nossa abordagem.

2.4 Conclusões

Neste capítulo, apresentamos uma visão geral da fundamentação teórica desta dissertação, discutindo os principais assuntos relacionados à nossa abordagem. Para isso, descrevemos o problema de OBDA e, uma vez que propomos resolver este problema de maneira modular, ou seja, dividindo-o em atividades independentes, discorremos sobre as cada uma destas atividades. Apresentamos uma revisão bibliográfica acerca dos trabalhos existentes para a tradução dos modelos relacional e XML para ontologias, bem como das estratégias existentes para transformação de consultas SPARQL para SQL e *XQuery*. Destacamos ainda quais trabalhos seriam mais adequados ao nosso propósito. Em seguida, abordamos alguns conceitos sobre mapeamento entre ontologias.

O problema de reescrita de consultas, que consiste no problema chave desta dissertação, também foi discutido. Exibimos os principais cenários de aplicação para reescrita, em particular, utilizando ontologias, e descrevemos as abordagens existentes que lidam com este problema. Discorremos, de maneira mais detalhada, sobre os trabalhos que possuem características mais próximas ao nosso e, por fim, realizamos uma análise comparativa entre estes trabalhos. O próximo capítulo apresenta o ambiente de OBDA utilizado, enquanto o Capítulo 4 descreve a estratégia de reescrita desenvolvida.

Tabela 2.7. Resumo das Abordagens para Reescrita Relacionadas

Abordagem	Objetivo	Modelo (Linguagem) de Representação das Ontologias	Linguagem de Consulta	Tipo de Mapeamento/ Correspondência	Método de Reescrita
What to Ask a Peer **[Calvanese et al. 2004]**	Definir formalmente o seguinte problema: dado duas ontologias, como responder a uma consulta submetida sobre uma ontologia alvo dado um conjunto de mapeamentos entre estas ontologias.	Base de Conhecimento em Lógica de Primeira Ordem (LPO)	Consulta em LPO	Homogêneo/ Mapeamentos GAV expresso em um fragmento de LPO sem considerar funções e operações de comparação/ Relações de equivalência	Reformula a consulta submetida sobre um *peer* alvo utilizando restrições (subsunção, tipagem e funcionalidade) da sua própria base de conhecimento. Em seguida, usa os mapeamentos para reescrever a consulta sobre o *peer* fonte.
SomeRDF [Adjiman et al. 2007]	Reescrita de consultas entre *peers* na Web Semântica	Core-RDF	Consulta em LPO	Homogêneo/Expresso em LPO sem considerar funções e operações de comparação/ Relações de equivalência, inclusão e disjunção	Transforma a consulta expressa em LPO para lógica proposicional e reduz a resposta à consulta ao problema de encontrar as conseqüências (*consequence finding*) de uma certa fórmula, na teoria da lógica proposicional, aplicado a um cenário distribuído.
SemRef [Fernandes 2009]	Reformulação de consultas em ambientes dinâmicos e distribuídos e em OPDMS. Possibilita geração de consultas aproximadas, além das exatas.	OWL	ALC - DL/ SPARQL	Homogêneo/ Correspondências expressas em DDL, sem funções e operações de comparação/ Relações de equivalência, especialização, generalização, proximidade, disjunção e agregação (parte-de; conjunto-de)	Utiliza o conjunto de correspondências, informações de contextuais, incluindo as preferências do usuário, para gerar consultas exatas e/ou aproximadas.

[Poggi et al. 2008; Calvanese et al. 2009]	Acesso a dados (OBDA) relacionais através de uma ontologia de domínio, utilizando serviços de inferência e reescrita de consultas.	OWL (DL $Lite_A$)	Consulta em LPO	Homogêneo/ Mapeamentos GAV entre um conceito na ontologia e uma consulta em SQL, sem operações de comparação / Relações de equivalência	Reformula a consulta submetida sobre a ontologia utilizando restrições da ontologia. Em seguida, usa os mapeamentos para reescrever a consulta sobre a base de dados relacional.
[Correndo et. al 2010]	Reescrita de consultas SPARQL aplicada aos Linked Data.	RDF/OWL	SPARQL	Homogêneo/ Expresso em RDF, sem considerar funções ou operações de comparação/ Relações de equivalência	Extrai o padrão de grafo da consulta SPARQL (sem considerar o operador de filtro) e reescreve cada padrão de tripla de acordo com os mapeamentos.
[Markis et. al 2010]	Reescrita de consultas SPARQL entre duas ontologias. A idéia é aplicar a abordagem em sistemas baseados em mediadores para integração de dados RDF.	RDF/OWL	SPARQL	Homogêneo/ Expresso em DL, sem funções e com operações de comparação sobre a ontologia fonte/ Relações de equivalência e subsunção	Reescreve cada padrão de tripla considerando que estes podem ser padrões de tripla de dados ou de esquema.
SQuOL	Reescrita de consultas entre ontologias com estruturas distintas. Pode ser aplicada em OBDA e em cenários de integração.	RDF/OWL	SPARQL	Heterogêneo/ Expresso em uma extensão de *Datalog*, com funções e operações de comparação sobre a ontologia fonte e alvo/ Relações de equivalência e subsunção entre classes	Utiliza um método que combina noções de programação em lógica com a semântica de SPARQL. Considera as funções, contexto nas regras e operadores de comparação durante a reescrita. Descarta algumas subconsultas que retornariam vazio.

CAPÍTULO 3
Um Ambiente para Acesso a Dados Baseado em Ontologias

Este capítulo apresenta o ambiente para Acesso a Dados Baseado em Ontologias utilizado neste trabalho. Inicialmente, Na Seção 3.1, são discutidos alguns dos cenários através dos quais este tipo de acesso pode ser realizado. Em seguida, é apresentada a arquitetura do ambiente, com a descrição dos principais módulos deste. Na Seção 3.2, os mapeamentos utilizados neste trabalho são caracterizados, sendo descrito, caso a caso, o processo de geração destes mapeamentos. Além disso, tal seção exibe trechos de ontologias que serão utilizadas como ilustração, ao longo desta dissertação.

3.1 Descrição do Ambiente

3.1.1 Considerações Iniciais

O objetivo do ambiente para Acesso a Dados Baseado em Ontologias (*Ontology-Based Data Access* – OBDA) apresentado neste trabalho é prover uma infraestrutura que possa ser utilizada por aplicações que necessitem interagir com fontes de dados por meio de consultas submetidas sobre ontologias. Ao se especificar um ambiente deste tipo, é necessário considerar que ontologias e fontes de dados, além de sintaticamente diferentes, possuem níveis de abstração e expressividade distintos. Além disso, as ontologias não devem necessariamente refletir a estrutura do esquema da fonte, mas sim representar de maneira adequada o domínio dos dados. Dessa forma, pode se tornar necessário lidar com estruturas distintas entre os dois modelos, o que demanda a utilização de mapeamentos mais complexos, isto é, mapeamentos que não podem ser expressos através de associações simples entre termos [An et al. 2005]. No ambiente de OBDA aqui apresentado, tais mapeamentos serão utilizados tanto durante o processo de reescrita das consultas quanto para a reestruturação dos resultados destas.

Diante disso e visando levantar alguns pontos relevantes da abordagem adotada, inicialmente, discutiremos dois possíveis cenários, ilustrados nas Figuras 3.1(a) e 3.1(b),

para realizar acesso a fontes de dados a partir de ontologias. Tais cenários são compostos pelos seguintes elementos: uma ontologia de domínio (O_D), que representa a ontologia de referência utilizada para acessar a fonte de dados; o esquema da fonte de dados; uma ontologia local (O_L), que corresponde a uma tradução sintática do esquema da fonte para uma linguagem de representação de ontologias; e um conjunto de mapeamentos, os quais podem ser classificados como *homogêneos* ou *heterogêneos*. Consideramos que um mapeamento é *simples* se ele pode ser representado através de uma associação direta entre os termos envolvidos. Note que, em geral, mapeamentos heterogêneos não podem ser representados através de associações diretas. Finalmente, utilizamos o termo *objeto* para referenciar as instâncias de uma ontologia, ou seja, triplas da forma: *(sujeito, predicado, objeto).*

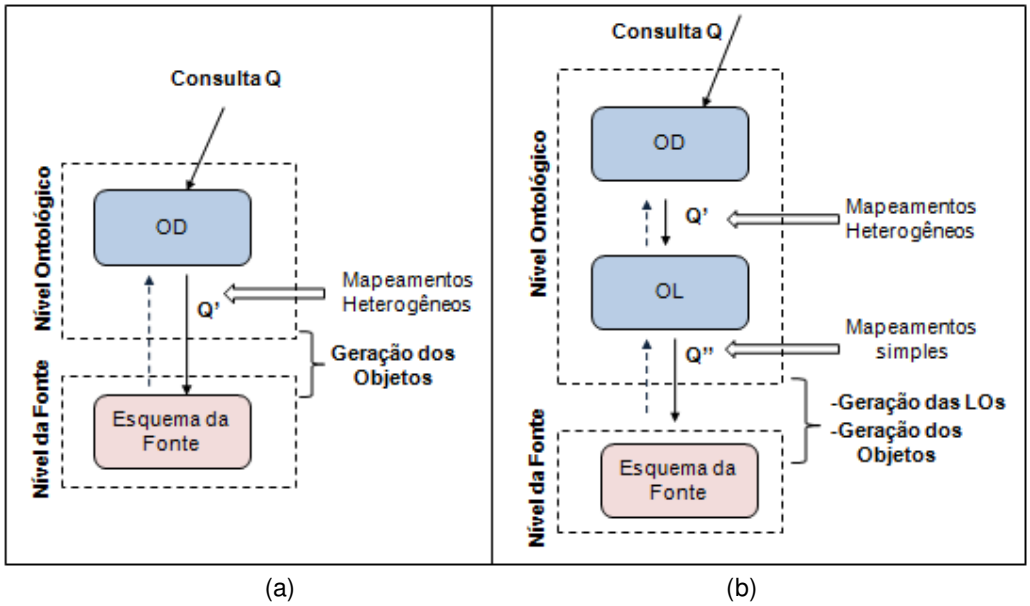


Figura 3.1. Cenários 1 e 2 para acesso a dados a partir de ontologias

Como pode ser observado na Figura 3.1, o problema de transitar entre o "mundo" das ontologias e do esquema fonte, reescrevendo consultas e reestruturando resultados, pode ser tratado sob duas perspectivas, representadas pelos Cenários 1 e 2. Dependendo da abordagem adotada, o ponto de maior dificuldade ao longo do processo irá variar, uma vez que tal ponto reside no local onde se encontram os mapeamentos mais complexos.

Cenário 1: A ontologia de domínio O_D é mapeada diretamente para o esquema da fonte de dados. Com isso, para reescrever uma consulta submetida sobre a O_D em termos de uma consulta sobre esta fonte, é necessário tratar duas questões em um

mesmo passo: (i) troca de modelo de dados e (ii) manipulação de mapeamentos heterogêneos.

Cenário 2: Inicialmente, é gerada uma ontologia local O_L a partir do esquema da fonte de dados, sendo O_L mapeada tanto com o esquema da fonte (mapeamentos simples) quanto com a ontologia de domínio O_D (mapeamentos heterogêneos). Observe que esta estratégia permite que os mapeamentos heterogêneos sejam alocados entre as ontologias, que estão descritas em um modelo de dados comum.

Na abordagem desenvolvida neste trabalho, focamos no **Cenário 2**, com o intuito de desenvolver uma solução mais genérica e modular, pois nos abstraímos do tipo de fonte de dados, dividindo o problema principal em dois subproblemas intermediários:

(i) reescrita e resolução de consultas entre uma ontologia fonte e uma ontologia alvo (O_L e O_D, respectivamente), as quais estão em um mesmo modelo de dados;

(ii) tradução de consultas entre modelos de dados distintos, porém com estruturas semelhantes, por meio do desenvolvimento de tradutores.

Observe que o subproblema (i) apresenta um grau de dificuldade maior que o (ii), devido à utilização de mapeamentos heterogêneos entre as ontologias. Além disso, uma vez que a ontologia local representa apenas uma tradução sintática do esquema da fonte de dados, a consulta reescrita sobre a ontologia fonte O_L pode ser facilmente mapeada para a base de dados correspondente.

A Figura 3.2 ilustra a arquitetura geral do mecanismo de OBDA proposto. Nela, é possível visualizar dois ambientes principais: um ambiente correspondente ao conjunto de módulos e atividades realizadas em *tempo de projeto* e outro ambiente com módulos e atividades realizadas em *tempo de execução*. A seguir, ambos ambientes serão descritos com maiores detalhes.

3.1.2 Ambiente de Tempo de Projeto

Este ambiente é composto pelos seguintes módulos:

- **Gerador da Ontologia Local:** tem como função a geração de uma ontologia (ontologia local - O_L) a partir dos elementos (esquema e/ou instâncias) da fonte de dados recebida como entrada. Além disso, gera como saída um conjunto de mapeamentos entre O_L e tal esquema da fonte ($M_{OL\text{-}EFD}$). Este módulo encontra-se

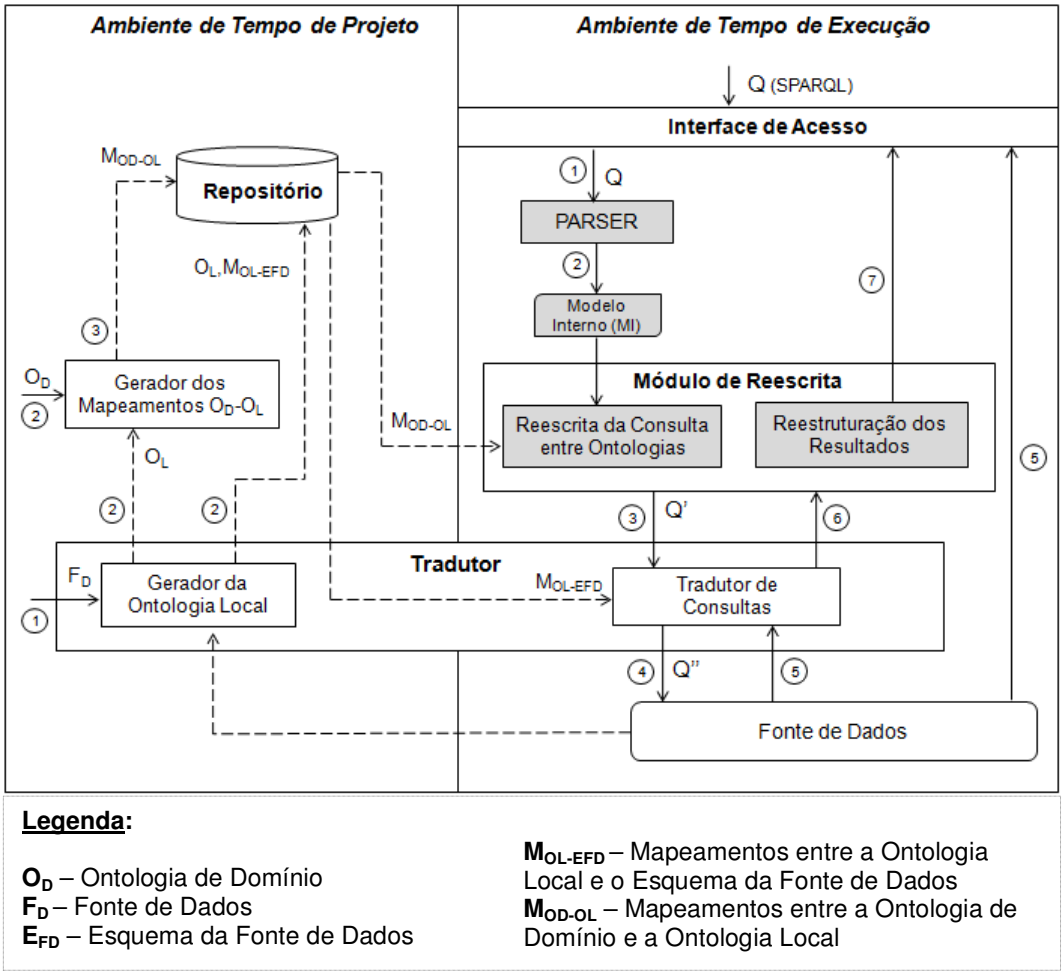


Figura 3.2. Arquitetura do mecanismo de OBDA proposto

inserido em um módulo maior, o **Tradutor**, o qual é responsável pela interação do sistema com a fonte de dados. Note que o **Tradutor** possui ainda outro submódulo, que funciona em tempo de execução, e que será detalhado mais adiante. Atualmente, existem diversos trabalhos para geração de ontologias a partir de bancos de dados relacionais e XML, conforme discutido no Capítulo 2, onde apontamos os mais apropriados a serem utilizados neste módulo.

- **Gerador dos Mapeamentos entre Ontologias (M_{OD-OL})**: recebe duas ontologias como entrada (O_D e O_L) e gera como saída um conjunto de mapeamentos entre elas. Maiores detalhes sobre este módulo serão dados na próxima seção.

- **Repositório:** armazena as ontologias e os mapeamentos gerados, disponibilizando-os para que os demais módulos do ambiente possam manipulá-los.

O funcionamento do mecanismo em tempo de projeto ocorre a partir do recebimento dos elementos da fonte de dados (F_D) e de uma ontologia (O_D), através da

qual a fonte deve ser registrada e acessada. Com esta entrada recebida, os seguintes passos (representados pelos números na Figura 3.2) são executados:

1. A fonte de dados é enviada para o módulo de *Geração da Ontologia Local*.
2. Após a geração da O_L e dos respectivos mapeamentos $M_{OL\text{-}EFD}$, estes são armazenados no repositório, ao mesmo tempo em que a O_L é enviada ao módulo de *Geração dos Mapeamentos O_D-O_L*.
3. Uma vez gerados, os mapeamentos $M_{OD\text{-}OL}$ são armazenados no repositório.

3.1.3 Ambiente de Tempo de Execução

Após a realização das atividades em tempo de projeto, o ambiente encontra-se preparado para a realização das consultas, em tempo de execução. As maiores contribuições deste trabalho encontram-se nos módulos inseridos neste ambiente. Tais módulos, os quais estão vinculados à atividade de reescrita, aparecem destacados na Figura 3.2 e terão suas atividades detalhadas no próximo capítulo.

- ***PARSER*:** é responsável por analisar a semântica da consulta, identificando seus conceitos, operadores e objetivos (questionamento da consulta). Neste módulo, são extraídos os principais componentes da consulta recebida, os quais são então mapeados para um modelo interno que será manipulado no ambiente ao longo do processo de reescrita.

- **Reescrita da Consulta entre Ontologias:** módulo responsável pelas funcionalidades vinculadas a um dos principais questionamentos deste trabalho: dado duas ontologias (O_D e O_L) estruturalmente diferentes e um conjunto de mapeamentos heterogêneos ($M_{OD\text{-}OL}$) entre elas, como reescrever uma consulta submetida sobre a ontologia alvo (O_D) em uma consulta na ontologia fonte (O_L)?

- **Reestruturação dos Resultados:** uma vez que a consulta original foi submetida sobre a ontologia de domínio, devemos fornecer a possibilidade de apresentar o resultado desta consulta conforme tal ontologia. Para isso, os dados obtidos como resultado da consulta original devem ser rearranjados em um processo semelhante ao problema de *data exchange* [Fagin et.al 2009]. Neste tipo de problema, um conjunto I_s de instâncias pertencentes a um esquema fonte S deve ser transformado em um conjunto I_T de instâncias pertencentes a um esquema alvo T. Esta transformação deve ser realizada respeitando tanto os mapeamentos M_{ST} entre os esquemas quanto as restrições do esquema alvo T.

- **Tradutor de Consultas:** desempenha as atividades em tempo de execução do módulo **Tradutor**. Para isso, com base nos mapeamentos $M_{OL\text{-}EFD}$, traduz as consultas submetidas sobre a O_L para consultas sobre a fonte de dados, considerando o problema da troca de modelos. Além disso, retorna os resultados como instâncias das ontologias**.** Trabalhos discutidos no Capítulo 2 podem ser reaproveitados aqui.

A arquitetura proposta prevê que a consulta seja recebida a partir de uma **Interface de Acesso**, que pode ser representada por uma interface gráfica ou por uma aplicação qualquer. De posse da consulta sobre a ontologia de domínio (O_D), as seguintes atividades são realizadas (representadas pelos números na Figura 3.2):

1. Inicialmente, a consulta Q é recebida pelo ***PARSER***, que a converte para o modelo interno.
2. Em seguida, Q é enviada para o módulo de **Reescrita de Consultas entre Ontologias**, sendo gerada uma nova consulta Q' sobre a O_L.
3. A consulta Q' é enviada ao módulo **Tradutor de Consultas**. Para que a esta consulta possa ser traduzida em termos do esquema da fonte de dados, são também utilizados como entrada para este módulo os mapeamentos $M_{OL\text{-}EFD}$. A partir de então, é gerada a consulta Q'', chamada de *consulta local*, na linguagem do esquema da fonte de dados.
4. A consulta Q'' é executada sobre a fonte de dados.
5. Finalmente, após a execução da consulta, os dados recuperados podem ser fornecidos à **Interface de Acesso**. Esse processo pode ocorrer de duas maneiras, dependendo dos requisitos da aplicação: (i) o resultado é devolvido na forma de instâncias do modelo de dados da fonte, como por exemplo, através de tuplas relacionais ou como um documento XML; (ii) considerando que a consulta inicial foi realizada sobre uma ontologia, pode-se esperar que o usuário deseje receber os dados como instâncias desta ontologia. Neste último caso, são necessários passos adicionais, representados na figura como (6) e (7), onde os dados oriundos da fonte são reconstruídos como objetos da ontologia na qual a consulta foi submetida (O_D).

É válido destacar que os dados a serem consultados nas fontes podem estar disponíveis no modelo relacional, XML ou em RDF. Neste último caso, não há a necessidade da utilização dos tradutores.

3.2 Caracterização dos Mapeamentos entre as Ontologias

Como abordado anteriormente, a principal contribuição do nosso trabalho consiste na reescrita de consultas entre duas ontologias, problema que será detalhado no capítulo seguinte. No entanto, para que esta reescrita seja possível, é necessário caracterizarmos os mapeamentos que serão manipulados durante esta atividade. Nesta seção, descreveremos a lei de formação dos mapeamentos entre as ontologias, apresentando o processo de geração destes, o qual é desempenhado pelo *módulo gerador dos mapeamentos*.

Em geral, um *mapeamento entre* uma ontologia *fonte* O_S e uma ontologia *alvo* O_T é caracterizado por um conjunto de expressões que definem os conceitos de O_T em termos dos conceitos de O_S, de forma que estes sejam semanticamente correspondentes [Leme 2009]. Segundo [Euzenat and Shvaiko 2007], um mapeamento é ainda uma versão orientada (ou direcionada) de um alinhamento, representada através de algum formalismo.

Neste trabalho, os mapeamentos são expressos através de um conjunto de regras, as quais são obtidas conforme a estratégia proposta em [Sacramento et. al 2010a] e descrita nas seções subseqüentes. Tal estratégia divide-se em duas etapas principais: (i) *matching de vocabulários*, que define um modelo de *matching* para representação do alinhamento entre as entidades de duas ontologias distintas; e (ii) *geração das regras de mapeamento*, que induz um conjunto de regras a partir do modelo obtido na etapa (i). A Seção 3.2.3 apresenta a etapa (i). A Seção 3.2.4 descreve o formalismo de regras que utilizamos para a representação dos mapeamentos e, finalmente, a Seção 3.2.5 discute a etapa (ii).

Antes de prosseguirmos com a definição dos mapeamentos, iremos apresentar, na Seção 3.2.1, um dialeto de OWL, denominado OWL *Extralite* [Leme 2009], no qual estão representadas as ontologias utilizadas neste trabalho. Em seguida, na seção 3.2.2, são exibidos exemplos com trechos das ontologias que utilizaremos para ilustrações, durante o restante deste capítulo.

3.2.1 OWL *Extralite*

OWL *Extralite* é um dialeto de OWL que contém os construtores fundamentais presentes na maioria das linguagens para representação de ontologias, bem como nos

modelos de dados conceituais, tais como UML. Assim, uma ontologia *OWL Extralite* é um par $O = (V, A)$, tal que:

- ***V*** representa o *vocabulário* da ontologia, composto por classes, propriedades de tipo de dado e propriedades de objeto definidas na ontologia.
- ***A*** representa um conjunto de *axiomas*, correspondentes às restrições definidas sobre *O*, as quais são:
  - **Restrições sobre classes:** subsunção (*rdfs:subclassOf*), disjunção (*owl:disjointWith*) e equivalência (*owl:equivalentClass*);
  - **Restrições sobre propriedades:** cardinalidades mínima e máxima (owl:*minCardinality* e *owl:maxCardinality*), domínio (*rdfs:domain*) e *range (rdfs:range)*.

Como característica das propriedades, *OWL Extralite* admite a utilização dos seguintes construtores: *owl:inverseFunctionalProperty*, que captura a representação de chaves simples; *owl:inverseOf*, que indica o relacionamento inverso entre propriedades; e *owl:functionalProperty*, que determina que uma propriedade deve possuir, no máximo, um valor para cada objeto. Além disso, este dialeto suporta a definição de indivíduos (instâncias na ontologia).

Note que um esquema *OWL Extralite* é uma coleção de triplas RDF que utilizam um subconjunto do vocabulário OWL. Em comparação com os dialetos de OWL amplamente conhecidos (*Lite*, *DL* e *Full*), a maioria dos construtores do *Extralite* advém do *OWL Lite*, exceto por *owl:disjointWith* e *owl:inverseFunctionalProperty.* O primeiro faz parte do dialeto *OWL DL* e o segundo, quando aplicado a propriedades de tipo de dado, é suportado somente por *OWL Full*.

Finalmente, para referências posteriores, definimos que uma classe c domina uma classe d em uma ontologia O [Leme 2009] se, e somente se, $c = d$ ou existe uma sequência $(c_1,c_2,...,c_n)$ tal que $c = c_1$, $d = c_n$ e c_{n-1} é uma superclasse de c_n e, para cada $i \in [1,n-2)$, temos:

- c_{i+1} e c_i são classes e c_{i+1} é declarada como uma subclasse de c_i em O ou
- c_{i+1} é uma propriedade de objeto em O, a qual o domínio é c_i ou
- c é uma propriedade de objeto em O, a qual o *range* é c_{i+1}.

Dizemos ainda que $\pi = (c_1, c_2,..., c_n)$ é um *caminho de dominância* (*dominance path*) de c para d e que $\theta = (pk_1, pk_2,..., pk_n)$, uma subseqüência de π consistindo das

propriedades de objeto que ocorrem em π é o *caminho de propriedades* (*property path*) correspondente a π. É válido destacar que θ pode ser uma seqüência vazia.

3.2.2 Ontologias Utilizadas nos Exemplos

As Figuras 3.3, 3.4 e 3.5 ilustram trechos de ontologias (O_1, O_2 e O_3, respectivamente), cujas diferenças estruturais serão exploradas durante o processo de caracterização dos mapeamentos, bem como durante a etapa de reescrita das consultas. Tais ontologias pertencem ao domínio de Vendas e, a respeito delas, consideramos que:

- O_1 representa uma ontologia de domínio fornecida pela aplicação; O_2 e O_3 são ontologias que modelam os esquemas das lojas virtuais *Amazon* e *eBay*, respectivamente. Dessa forma, temos que a ontologia O_1 desempenhará o papel de ontologia *alvo* e as demais, serão as ontologias *fontes*.
- Os prefixos "*v:*", "*a:*" e "*e:*" representam os *namespaces* referentes ao vocabulário das ontologias O_1 (Vendas), O_2 (*Amazon*) e O_3 (*eBay*), respectivamente.
- Visando facilitar o entendimento, todas as ontologias estão expressas na notação UML e, embora não esteja indicado nas figuras, assumimos que:
  - Todas as propriedades de tipo de dado possuem *maxCardinality* igual a 1, ou seja, possuem um único valor.
  - Na ontologia O_1, as propriedades *v:titulo*, *v:nomeEditora*, *v:isbn* e *v:nomeGravadora* são *inversas funcionais*, o que indica que elas constituem chaves simples para as classes *v:Produto*, *v:Editora*, *v:Publicacao* e *v:Gravadora,* respectivamente.
  - Em O_2, as propriedades inversas funcionais são: *a:descricao* e *a:isbn*, as quais representam chaves simples para as classes *a:Produto* e *a:Livro*, respectivamente.
  - Em O_3, *e:descricao* e *e:nome* são as propriedades inversas funcionais, o que indica que elas representam chaves para as classes *e:Produto* e *e:Vendedor*, respectivamente.

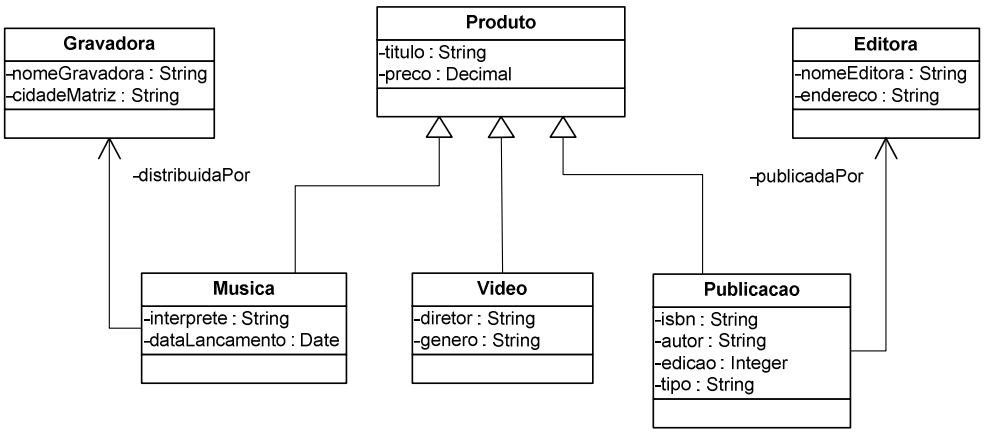


Figura 3.3. Trecho da ontologia de domínio de Vendas O_1

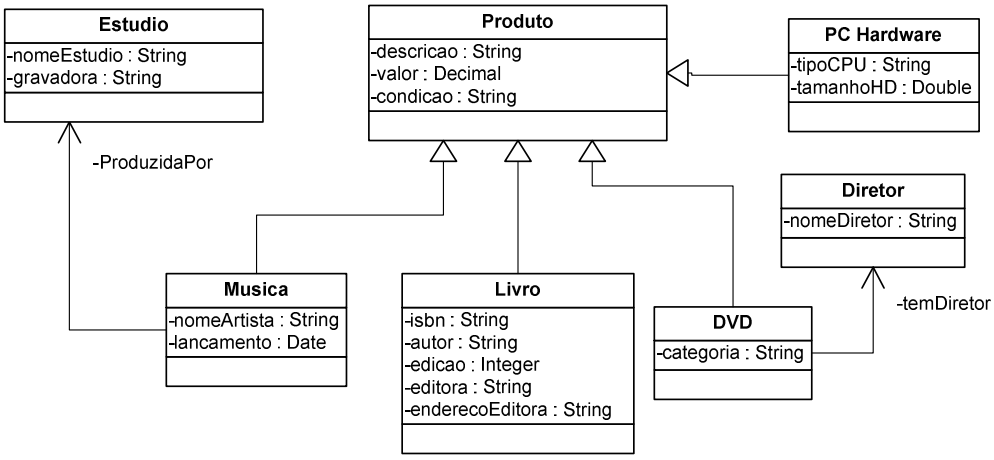


Figura 3.4. Trecho da Ontologia da *Amazon* O_2

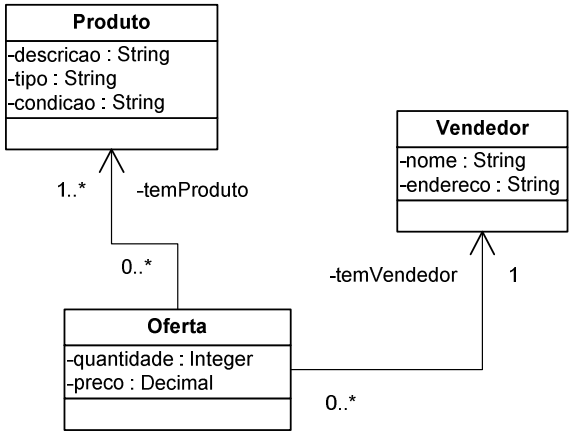


Figura 3.5. Trecho da Ontologia do *eBay* O_3

3.2.3 *Matching* de Vocabulários

Seja O_S uma ontologia *fonte* e O_T uma ontologia *alvo*, tal que V_S e V_T representam seus respectivos vocabulários. Sejam ainda C_S e C_T os conjuntos de classes, e P_S e P_T os conjuntos de propriedades em V_S e V_T, respectivamente. Um *matching* contextualizado de vocabulários [Sacramento et al. 2010a] entre O_S e O_T é um *matching* cujas correspondências obtidas podem ser representadas através de um conjunto finito S de sêxtuplas (v_1, e_1, r_1, v_2, e_2, r_2), tal que:

- Se (v_1, v_2) ∈ C_S x C_T, então e_1 e e_2 são a classe topo $\top$ e r_1 e r_2 são restrições que delimitam um subconjunto das classes v_1 e v_2, respectivamente;
- Se (v_1, v_2) ∈ P_S x P_T, então e_1 e e_2 são classes em C_S e C_T, as quais devem ser subclasses dos domínios, os próprios domínios ou os domínios *restritos* das propriedades v_1 e v_2, respectivamente. Já r_1 e r_2 são restrições que delimitam um subconjunto das classes e_1 e e_2, respectivamente.

Intuitivamente, dizemos que, no primeiro caso, a sêxtupla (v_1, $\top$, r_1, v_2, $\top$, r_2) indica que toda tripla (x, *rdf:type*, v_1), na qual x é um indivíduo de v_1 que satisfaz a restrição r_1, corresponde à tripla (x, *rdf:type*, v_2), onde x é um indivíduo de v_2, que satisfaz r_2. Já no segundo caso, a sêxtupla (v_1, e_1, r_1, v_2, e_2, r_2) indica que toda tripla (x, v_1, a), onde x é um indivíduo de e_1 que satisfaz r_1, corresponde à tripla (x, v_2, a), onde x é um indivíduo de e_2 que satisfaz r_2. Em ambas as triplas, a é o valor que as propriedades (v_1 ou v_2) assumem para o indivíduo x.

Dessa forma, temos que se (v_1, e_1, r_1, v_2, e_2, r_2) ∈ S, então S *alinha* v_1 e v_2 no contexto de (e_1,r_1) e (e_2,r_2), respectivamente. Isto significa que (e_i, r_j) é o *contexto* de v_i e (v_i,e_i,r_i) é um *conceito contextualizado*, para i = 1, 2. Note que, neste modelo para especificação de *matching*, o contexto indica as condições necessárias para que dois conceitos de ontologias distintas sejam devidamente alinhados.

No que diz respeito à definição das restrições, consideramos que uma restrição r é uma tripla do tipo (*prop, op, const*), onde: *prop* é uma propriedade de tipo de dados, *op* é uma operação de comparação e *const* é um valor constante atribuído à propriedade *prop*. Tal valor é utilizado, juntamente com o operador de comparação *op*, com o intuito de estabelecer um filtro que determine um subconjunto da classe sobre a qual a restrição está sendo aplicada. Os operadores de comparação considerados neste trabalho são: igual (=), maior (>), maior ou igual (>=), menor (<), menor ou igual (<=) e diferente

($\neq$). O operador de igualdade pode ser aplicado tanto sobre *strings* quanto sobre valores numéricos, enquanto os demais operadores são permitidos somente sobre valores numéricos.

Finalmente, definimos que uma classe v_1 (em O_S) é *semanticamente relacionada* a uma classe v_2 (em O_T), tal relação é representada por ($v_1 \sqsubseteq^{sr} v_2$), se [Sacramento et al. 2010a]:

- $(v_1, \top, r_1, v_2, \top, r_2) \in S$;
- $(v_1, \top, r_1, v_2', \top, r_2) \in S$, onde v_2' é uma subclasse de v_2.

Caso contrário, definimos que v_1 não é semanticamente relacionada a v_2 e esta relação é representada por $v_1 \not\sqsubseteq^{sr} v_2$.

As Tabelas 3.1 e 3.2 ilustram o conjunto de sêxtuplas obtido a partir do *matching* realizado entre as ontologias (fonte e alvo) de nosso exemplo, isto é, entre O_2 e O_1 e entre O_3 e O_1, respectivamente. Note que as restrições podem possuir o valor $\top$ numa sêxtupla, o que indica a ausência de restrições sobre suas respectivas classes.

Na Tabela 3.1, a sêxtupla da linha 1 indica que as propriedades *a:descricao* e *v:titulo* se alinham no contexto das classes *a:Musica* e *v:Musica*, respectivamente. Para estas classes há também uma correspondência, exibida na linha 4. Observe, ainda nas linhas 1 e 4, que as colunas referentes às restrições estão preenchidas com a classe topo $\top$, ao contrário, por exemplo, da sêxtupla apresentada na linha 7. Nesta última, as propriedades são alinhadas somente quando a publicação (*v:Publicacao*), na ontologia *alvo* O_1, for do tipo *livro* (*v:tipo* = "livro"). Já na linha 5, temos uma correspondência entre as propriedades *a:nomeDiretor* e *v:diretor*, embora as classes pertencentes aos seus respectivos contextos não estejam semanticamente relacionadas. Casos semelhantes ocorrem nas sêxtuplas das linhas 6, 13 e 14. Em situações como estas, torna-se necessário buscar, na ontologia fonte ou na ontologia alvo, um *caminho* θ que permita relacionar os contextos, de forma que as instâncias presentes nas ontologias possam ser corretamente associadas. A caracterização deste caminho, apresentada na Seção 3.2.1 (caminho de dominância), é importante para que as regras de mapeamento entre os conceitos das ontologias possam ser geradas de maneira correta, conforme veremos posteriormente. Como ilustração, é possível observar, nas ontologias O_1 e O_2 (Figuras 3.3 e 3.4, respectivamente), um caminho θ, na ontologia fonte, composto pela propriedade de objeto *a:temDiretor*. Este caminho permite que, a partir de duas classes

semanticamente relacionadas (*v:Video* e *a:DVD)*, seja possível alcançar a propriedade *v:nomeArtista*.

Tabela 3.1. *Matching* de Vocabulários entre a ontologia da *Amazon* O_2 e a ontologia de Vendas O_1

#	Ontologia da *Amazon* O_2			Ontologia de Vendas O_1		
	v_1	e_1	r_1	v_2	e_2	r_2
1	a:descricao	a:Musica	⊤	v:titulo	v:Musica	⊤
2	a:valor	a:Musica	⊤	v:preco	v:Musica	⊤
3	a:lancamento	a:Musica	⊤	v:dataLancamento	v:Musica	⊤
4	a:Musica	⊤	⊤	v:Musica	⊤	⊤
5	a:nomeArtista	a:Musica	⊤	v:interprete	v:Musica	⊤
6	a:gravadora	a:Estudio	⊤	v:nomeGravadora	v:Gravadora	⊤
7	a:descricao	a:Livro	⊤	v:titulo	v:Publicacao	v:tipo= “livro”
8	a:valor	a:Livro	⊤	v:preco	v:Publicacao	v:tipo= “livro”
9	a:isbn	a:Livro	⊤	v:isbn	v:Publicacao	v:tipo= “livro”
10	a:autor	a:Livro	⊤	v:autor	v:Publicacao	v:tipo= “livro”
11	a:edicao	a:Livro	⊤	v:edicao	v:Publicacao	v:tipo= “livro”
12	a:Livro	⊤	⊤	v:Publicacao	⊤	v:tipo= “livro”
13	a:editora	a:Livro	⊤	v:nomeEditora	v:Editora	⊤
14	a:enderecoEditora	a:Livro	⊤	v:endereco	v:Editora	⊤
15	a:descricao	a:DVD	⊤	v:titulo	v:Video	⊤
16	a:valor	a:DVD	⊤	v:preco	v:Video	⊤
17	a:categoria	a:DVD	⊤	v:genero	v:Video	⊤
18	a:nomeDiretor	a:Diretor	⊤	v:diretor	v:Video	⊤
19	a:DVD	⊤	⊤	v:Video	⊤	⊤

Na Tabela 3.2, podemos destacar a sêxtupla exibida na linha 2, na qual há uma restrição aplicada (*e:tipo* = “livro”) sobre a classe da ontologia fonte (*e:Produto*).

Tabela 3.2. Matching de Vocabulários entre a ontologia do *eBay* O_3 e a ontologia de Vendas O_1

#	Ontologia do eBay O_3			Ontologia de Vendas O_1		
	v_1	e_1	r_1	v_2	e_2	r_2
1	e:descricao	e:Produto	e:tipo= “livro”	v:titulo	v:Publicacao	v:tipo= “livro”
2	e:Produto	⊤	e:tipo= “livro”	v:Publicacao	⊤	v:tipo= “livro”
3	e:descricao	e:Produto	e:tipo= “musica”	v:titulo	v:Musica	⊤
4	e:Produto	⊤	e:tipo= “musica”	v:Musica	⊤	⊤

É importante salientar que as correspondências obtidas através das abordagens existentes para *matching* nem sempre são precisas e completas. Dessa forma, em muitos

casos, há a necessidade de se realizar uma etapa adicional, denominada de *pós-matching* [Sacramento et al. 2010a]. Esta etapa pode requerer a participação do usuário, com o intuito de validar os alinhamentos obtidos, bem como gerar novos alinhamentos, que não foram capturados automaticamente. No caso da abordagem aqui apresentada, é durante o *pós-matching* que os caminhos θ, citados anteriormente, são determinados.

3.2.4 Formalismo de Regras para Representação dos Mapeamentos

Uma vez que trabalhamos com ontologias estruturalmente distintas, necessitamos de um formalismo que possibilite a representação de *mapeamentos heterogêneos*. Atualmente, diversos trabalhos [Fernandes 2009; Makris et al. 2010] lidam apenas com *mapeamentos homogêneos*, representando-os, geralmente, através de Lógica Descritiva (*Description Logic – DL*). No entanto, embora *DL* seja útil para expressar relacionamentos entre conceitos (tais como subsunção e disjunção), esta linguagem apresenta algumas limitações em aspectos que são fundamentais para este trabalho. Por exemplo, no que tange à reestruturação de informações, *DL* não fornece meios para definição explícita de identificadores de objetos (*OIDs*). Visando contornar estas limitações, representamos nossos mapeamentos por meio de um conjunto de regras, conforme descrito a seguir.

Seja um átomo uma expressão do tipo $p(t_1,\ldots, t_n)$, onde p é um símbolo de predicado e t_i são termos, tal que $i=1$ até n. Um termo pode ser uma constante, uma variável ou uma função n-ária aplicada sobre outros termos. Sejam O_T e O_S duas ontologias e R uma linguagem de regras. Um mapeamento entre ontologias é especificado através de um conjunto de regras de mapeamento, cada uma das quais com a seguinte forma:

$$\Psi(w) \leftarrow \varphi_1(\mathrm{t}_1), \ldots, \varphi_\mathrm{n}(\mathrm{t}_\mathrm{n}) \text{ , onde:}$$

- $\varphi_1(t_1), \ldots, \varphi_n(t_n)$, chamado de *corpo* da regra, é um átomo ou uma conjunção de átomos, onde $\varphi_i(\mathrm{t_i})$ é um conceito ou um relacionamento pertencente à ontologia fonte O_S ou, ainda, uma operação de comparação e t_i é uma seqüência de termos.
- $\Psi(w)$, chamado de *cabeça* da regra, é um átomo, onde pode ser um conceito ou um relacionamento na ontologia alvo O_T e w é uma seqüência de termos.

Mais especificamente, o formalismo de regras que utilizamos é uma extensão de *Datalog* que possibilita a construção explícita de *OIDs* [Hull and Yoshikawa 1990]. Com isso, é possível a definição de *Skolem functions*, as quais são termos da forma

$f(t_1,...,t_n)$, onde f é um símbolo funcional de aridade $n>0$ e $t_1,...,t_n$ são termos pertencentes à O_S. Neste trabalho, estas funções são utilizadas para a geração de *URIs*, correspondentes aos objetos criados nas classes de O_T, a partir de uma ou mais propriedades da O_S.

3.2.5 Geração das Regras de Mapeamento

Nesta seção, será explanado como as regras de mapeamento são derivadas a partir das sêxtuplas discutidas anteriormente. De maneira geral, a geração de tais regras é guiada por uma série de casos, apresentados em [Sacramento et. al 2010], que podem ser divididos em dois grupos principais: geração de regras para as classes e geração de regras para as propriedades. Em ambas as situações, para cada entidade mapeada, consideramos as implicações em duas circunstâncias: (i) quando seus respectivos contextos estão alinhados e (ii) quando tais contextos não estão alinhados. Além disso, em cada caso enunciado, há um exemplo ilustrativo que referencia as ontologias e os alinhamentos apresentados nas Seções 3.2.2 e 3.2.3, respectivamente.

Seja S um *matching* contextualizado de vocabulários entre uma ontologia fonte O_S e uma ontologia alvo O_T. Seja ainda $s = (v_1,e_1,r_1,v_2,e_2,r_2)$ uma sêxtupla pertencente a S. O conjunto M das regras de mapeamento entre O_S e O_T é derivado de acordo com os casos definidos a seguir.

Caso 1. Se $os{:}v_1$ e $ot{:}v_2$ são classes e $ot{:}r_2$ é a classe topo $\top$, ou seja, não há restrição aplicada sobre a classe $ot{:}v_2$ da ontologia alvo, então:

$M = \{ot{:}v_2(x) \leftarrow os{:}v_1(x), os{:}r_1(x)\} \cup$

$\{ot{:}u(x) \leftarrow os{:}v_1(x), os{:}r_1(x)$, para cada superclasse u de $ot{:}v_2\}$

A primeira regra indica que todo indivíduo pertencente à classe v_1 e que satisfaz à restrição r_1, em O_S, pode ser mapeado como um indivíduo da classe v_2, em O_T. Já a segunda regra propaga o mapeamento para as superclasses de v_2, uma vez que toda instância de v_2 é também uma instância de suas superclasses. Neste trabalho, nos referimos a esta segunda regra como um *mapeamento de expansão na hierarquia de classes*.

Exemplo 1. Considere as sêxtuplas $s_{1\text{-}4}$ = (*a:Musica,* $\top$*,* $\top$*,v:Musica,* $\top$*,* $\top$) e $s_{2\text{-}4}$ = (*e:Produto,* $\top$*, e:tipo=*"*musica*"*,v:Musica,* $\top$*,* $\top$), as quais são exibidas na linha 4 das Tabelas 3.1 e 3.2, respectivamente. Observe que em $s_{1\text{-}4}$ as classes estão alinhadas sem

que haja restrições sobre elas, enquanto em $s_{2\text{-}4}$ há uma operação de comparação (=) restringindo os valores para os quais a classe *e:Produto* participa do alinhamento. Ambas as sêxtuplas se enquadram no Caso 1, portanto, os seguintes mapeamentos são gerados:

v:Musica(p) ← *a:Musica(p)*, para $s_{1\text{-}2}$

v:Musica(p) ← *e:Produto(p), e:tipo(p,k), k= "musica"*, para $s_{2\text{-}4}$

Como na ontologia alvo O_1 (Figura 3.3), *v:Musica* possui apenas uma superclasse (*v:Produto*), que é a sua classe pai imediata na hierarquia, as seguintes regras de mapeamentos são geradas, para cada uma das sêxtuplas deste exemplo:

v:Produto(p)← *a:Musica(p)* ($s_{1\text{-}4}$)

v:Produto(p)← *e:Produto(p), e:tipo(p,k), k= "musica"* ($s_{2\text{-}4}$)

Caso 2. Se $os{:}v_1$ e $ot{:}v_2$ são classes e $ot{:}r_2$ é uma restrição da forma $ot{:}p_2$ ■ $ot{:}c_2$, onde $ot{:}p_2$ é uma propriedade de tipo de dados, $do{:}c_2$ é uma constante e ■ é um dos seguintes operadores de comparação: =, >, >=, <, <=, ≠, então:

$M = \{ot{:}v_2(x) \leftarrow os{:}v_1(x), os{:}r_1(x)\}$ U

$\{ot{:}u(x) \leftarrow os{:}v_1(x), os{:}r_1(x)$, para cada superclasse *u* de $ot{:}v_2\}$ U

$\{ot{:}p_2(x,y) \leftarrow os{:}v_1(x), os{:}r_1(x)$, y $ot{:}c_2\}$

As duas primeiras regras são análogas ao Caso 1. Já a terceira regra é gerada com intuito de refletir a restrição aplicada sobre a classe $ot{:}v_2$, na ontologia alvo O_T.

Exemplo 2. A sêxtupla (*a:Livro,* ⊤*,* ⊤*, v:Publicacao,* ⊤*, v:tipo= "livro"*), exibida na linha 12 da Tabela 3.1, satisfaz ao Caso 2 e, por isso, induz a geração das seguintes regras:

v:Publicacao(p)← *a:Livro(p)*

v:Produto(p)← *a:Livro(p)*

v:tipo(p,y) ← *a:Livro(p), y= "livro"*

Neste exemplo, destacamos a terceira regra, a qual indica que toda instância da classe *a:Livro*, em O_S, corresponde a uma instância, de uma classe em O_T, cuja propriedade *v:tipo* possui o valor "*livro*".

Caso 3. Se $os{:}v_1$ e $ot{:}v_2$ são ambas propriedades de tipo de dado (ou ambas propriedades de objeto) tal que $os{:}e_1 \sqsubseteq^{sr} ot{:}e_2$, isto é, e_1 é semanticamente relacionada a e_2, conforme definido na Seção 3.2.3, então:

$M = \{ot{:}v_2(x,y) \leftarrow os{:}v_1(x,y), os{:}e_1(x), os{:}r_1(x)\}$

Esta regra indica que os valores da propriedade $os{:}v_1$ de um indivíduo x, em O_S, podem ser mapeados para a propriedade $ot{:}v_2$, em O_T, desde que x seja uma instância da classe $os{:}e_1$ e respeite a restrição $os{:}r_1$.

Exemplo 3. A sêxtupla (*e:descricao,e:Produto,e:tipo="musica",v:titulo,v:Musica,* $\top$), exibida na linha 3 da Tabela 3.2, induz a geração da seguinte regra:

v:titulo(p,y) $\leftarrow$ *e:descricao(p,y) e:Produto(p), e:tipo(p,k), k="musica"*

Nesta regra, temos que os valores da propriedade *e:descricao*, em O_3 (Figura 3.5), podem ser mapeados para a propriedade *v:titulo*, em O_1 (Figura 3.3), quando *p* for uma instância da classe *e:Produto* com o valor da propriedade *e:tipo* igual "*musica*".

Caso 4. Se $os{:}v_1$ e $ot{:}v_2$ são ambas propriedades de tipo de dado (ou ambas propriedades de objeto) tal que $os{:}e_1 \not\sqsubseteq^{sr} ot{:}e_2$ e há um caminho de propriedades $\theta = os{:}pk_1(x,x_1), os{:}pk_2(x,x_2),..., os{:}pk_n(x_{n-1},z)$, na ontologia fonte O_S, então:

$M = \{ot{:}v_2(x,y) \leftarrow os{:}pk_1(x,x_1), os{:}pk_2(x,x_2), ..., os{:}pk_n(x_{n-1},z),$
$os{:}v_1(z,y), os{:}e_1(z), os{:}r_1(z)\}$

Exemplo 4. A sêxtupla (*a:nomeDiretor, a:Diretor,* $\top$*, v:diretor, v:Video,* $\top$), exibida na linha 18 da Tabela 3.1, se enquadra no Caso 4. Portanto, a seguinte regra é gerada:

v:diretor(p,y) $\leftarrow$ *a:temDiretor(p,z), a:nomeDiretor(z,y) e:Diretor(z)*

Observe que esta regra não mapeia diretamente um valor de *a:nomeDiretor*, na ontologia fonte, para *v:diretor*, na ontologia alvo, embora estas propriedades estejam alinhadas. Isto acontece porque, caso tal mapeamento direto fosse gerado, estaríamos indicando que uma instância de *a:Diretor* poderia ser reinterpretada como uma instância de *v:Video*, classes estas que não estão semanticamente relacionadas. Dessa forma, torna-se necessária a definição de um caminho entre estas classes, de maneira que o mapeamento possa ser realizado corretamente. Neste exemplo, a propriedade *a:temDiretor* corresponde a este caminho, o que leva a geração de uma regra consistente, pois o contexto da referida propriedade é a classe *a:DVD*, a qual está alinhada com a classe *v:Video* (linha 19 da Tabela 3.1).

Caso 5. Se $os{:}v_1$ e $ot{:}v_2$ são ambas propriedades de tipo de dado (ou ambas propriedades de objeto) tal que $os{:}e_1 \sqsubseteq^{sr} ot{:}e_2$, mas há um caminho de propriedades θ, na ontologia alvo O_T, então:

M = {$ot{:}v_2$(f(y),y) ← $os{:}v_1$(x,y), $os{:}e_1$(x),$os{:}r_1$(x), onde $ot{:}v_2$ é uma propriedade de tipo de dados} U

{$ot{:}e_2$(f(y)) ← $os{:}v_1$(x,y), $os{:}e_1$(x),$os{:}r_1$(x), onde $ot{:}e_2$ é uma classe} U

{$ot{:}p_2$(x,f(y)) ← $os{:}v_1$(x,y), $os{:}e_1$(x),$os{:}r_1$(x), onde $ot{:}p_2$ é uma propriedade de objeto}

Este caso corresponde a uma situação na qual é preciso reestruturar informações ao se mapear elementos de *Os* para O_T. Por isso, a necessidade de se utilizar *skolem functions* que, neste trabalho, possibilitam a geração de novas *URIs* na ontologia alvo (O_T).

Em suma, a primeira regra mapeia as propriedades que estão alinhadas na sêxtupla, de maneira semelhante a outros casos descritos anteriormente. A diferença fica por conta da função *f*, a qual cria instâncias da classe $ot{:}e_2$, que está modelada somente na O_T, a partir do valor de uma ou mais propriedades da O_S. Já a segunda regra é responsável por mapear as triplas referentes à própria classe $ot{:}e_2$, enquanto a terceira regra mapeia as instâncias da propriedade de objeto na $ot{:}p_2$. É possível observar, através destas regras, que a reestruturação se caracteriza da seguinte forma: uma informação modelada na O_S através de uma classe encontra-se modelada na O_T por meio de várias classes e relacionamentos (propriedades de objeto) entre estas.

Exemplo 5. A sêxtupla (*a:editora, a:Livro,* $\top$*, v:nomeEditora, v:Editora,* $\top$), exibida na linha 13 da Tabela 3.1, enquadra-se no Caso 5. Portanto, as seguintes regras são geradas:

v:nomeEditora(fEditora(y),y) ← a:editora(p,y), a:Livro(p) (1)

v:Editora(fEditora(y)) ← a:editora(p,y), a:Livro(p) (2)

v:publicadaPor(p, fEditora(y)) ← a:editora(p,y), a:Livro(p) (3)

Neste exemplo, podemos observar que, em O_1, a informação referente a publicações encontra-se modelada através de duas classes e um relacionamento, ao passo que em O_2, temos apenas uma classe (*a:Livro)*. A informação disposta dessa maneira exige a utilização de mapeamentos com *skolem functions*. Isto porque há necessidade de se retornar como um objeto, correspondente a uma instância da classe

v:Editora, uma informação que estava originalmente modelada como uma propriedade do tipo *string* (*a:editora*). As regras 1, 2 e 3 ilustram os mapeamentos para esta situação. A regra 1 é derivada diretamente a partir da sêxtupla. As demais regras são obtidas a partir das ontologias, considerando o caminho θ. Nestas regras, *fEditora* é a função que gera as *URIs* em O_T e o parâmetro *y* é uma propriedade que identifica unicamente uma editora.

Caso 6. Se $os{:}v_1$ e $ot{:}v_2$ são ambas propriedades de tipo de dado (ou ambas propriedades de objeto) tal que $os{:}e_1 \overset{sr}{\sqsubseteq} ot{:}e_2$, mas há um caminho θ, na ontologia fonte O_S, bem como um caminho θ', na ontologia alvo O_T, então:

M = {$ot{:}v_2$(f(y),y) ← $os{:}v_1$(z,y),$os{:}e_1$(z), $os{:}r_1$(z), onde $ot{:}v_2$ é uma propriedade de tipo de dados} U

{$ot{:}e_2$(f(y)) ← $os{:}v_1$(z,y),$os{:}e_1$(z), $os{:}r_1$(z), onde $ot{:}e_2$ é uma classe} U

{$ot{:}p_2$(x,f(y)) ← $os{:}pc_1$(x,x_1),$os{:}pc_2$(x,x_2),...,$os{:}pc_n$(x_{n1},z),$os{:}v_1$(z,y), $os{:}e_1$(z), $os{:}r_1$(z), onde $ot{:}p_2$ é uma propriedade de objeto}

Este caso corresponde a uma combinação dos casos 4 e 5.

Exemplo 6. A sêxtupla (*a:gravadora,a:Estudio,* $\top$*, v:nomeGravadora, v:Gravadora,* $\top$), exibida na linha 6 da Tabela 3.1, induz a geração das seguintes regras:

v:nomeGravadora(fGravadora(y),y) ← a:gravadora(z,y), a:Estudio(z) (1)

v:Gravadora(fGravadora(y))← a:gravadora(z,y), a:Estudio(z) (2)

v:distribuidaPor(m,fGravadora(y))← a:interpretadaPor(m,z), a:gravadora(z,y), a:Estudio(z) (3)

Neste exemplo, o caminho na ontologia alvo O_1 é θ'= *v:distribuidaPor*,enquanto na ontologia fonte O_2 é θ = *a:produzidaPor*.

As Figuras 3.6 e 3.7 exibem as regras de mapeamento, agrupadas por classes e propriedades na ontologia alvo, induzidas a partir das Tabelas 3.1 e 3.2, respectivamente.

#1	v:Musica(m) ← a:Musica(m)
#2	v:Publicacao(p) ← a:Livro(p)
#3	v:Video(p) ← a:DVD(p)
#4	v:Produto(p) ← a:Musica(p)
#5	v:Produto(p) ← a:Livro(p)
#6	v:Produto(p) ← a:DVD(p)
#7	v:Editora(fEditora(y)) ← a:editora(p,y), a:Livro(p)
#8	v:Gravadora(fGravadora(y))← a:produzidaPor(m,z),a:gravadora(z,y), a:Estudio(z)
#9	v:tipo(p,y) ← a:Livro(p), y= “livro”
#10	v:interprete(m,y) ← a:nomeArtista(m,y)
#11	v:dataLancamento(m,y) ← a:lancamento(m,y), a:Musica(m)
#12	v:titulo(p,y) ← a:descricao(p,y), a:Musica(p)
#13	v:preco(p,y) ← a:valor(p,y), a:Musica(p)
#14	v:autor(p,y) ← a: autor(p,y), a:Livro(p)
#15	v:edicao(p,y) ← a:edicao(p,y), a:Livro(p)
#16	v:isbn(p,y) ← a: isbn(p,y), a:Livro(p)
#17	v:genero(p,y) ← a:categoria(p,y), a:DVD(p)
#18	v:titulo(p,y) ← a:descricao(p,y), a:Livro(p)
#19	v:preco(p,y) ← a:valor(p,y), a:Livro(p)
#20	v:titulo(p,y) ← a:descricao(p,y), a:DVD(p)
#21	v:preco(p,y) ← a:valor(p,y), a:DVD(p)
#22	v:diretor(p,y) ← a:temDiretor(p,z), a:nomeDiretor(z.y), a:DVD(p)
#23	v:nomeEditora(fEditora(y),y) ← a:editora(p,y), a:Livro(p)
#24	v:endereco(fEditora(y),z) ← a:enderecoEditora(p,z), editora(p,y), a:Livro (p)
#25	v:publicadaPor(p,fEditora(y)) ← a:editora(p,y), a:Livro(p)
#26	v:estilo(m,y) ← a:estilo(m,y), a:Musica(m)
#27	v:interprete(m,y) ← a:interpretadaPor(m,z), a:nomeArtista(z,y) e:Cantor(z)
#28	v:nomeGravadora(fGravadora(y),y) ← a:gravadora(z,y), a:Musica(z)
#29	v:distribuidaPor(m,fGravadora(y))← a:produzidaPor(m,z),a:gravadora(z,y), a:Musica(z)

Figura 3.6. Conjunto de Regras de Mapeamento entre a ontologia alvo O_1, de Vendas, e a ontologia fonte O_2, da *Amazon*

#1:	v:Musica(p) ← e:Produto(p), e:tipo(p,k), k= “musica”
#2:	v:Produto(p) ← e:Produto(p), e:tipo(p,k), k= “musica”
#3:	v:Publicacao(p) ← e:Produto(p), e:tipo(p,k), k= “livro”
#4:	v:Produto(p) ← e:Produto(p), e:tipo(p,k), k= “livro”
#5:	v:tipo(p,y) ← e:Produto(p), e:tipo(p,k), k= “livro”, y= “livro”
#6:	v:titulo(p,y) ← e:descricao(p,y), e:Produto(p), e:tipo(p,k), k= “livro”
#7:	v:titulo(p,y) ← e:descricao(p,y), e:Produto(p), e:tipo(p,k), k= “musica”

Figura 3.7. Conjunto de Regras de Mapeamento entre a ontologia alvo O_1, de Vendas, e a ontologia fonte O_3, do *eBay*

Observando as ontologias O_1, O_2 e O_3 (Figuras 3.3, 3.4 e 3.5, respectivamente) e os conjuntos de regras de mapeamentos obtidos, as seguintes observações podem ser realizadas:

(i) Alguns conceitos presentes na ontologia alvo O_1 não possuem um correspondente na ontologia fonte (O_2 ou O_3), isto é, não existe um mapeamento que relacione estes elementos. Como exemplo, podemos citar a propriedade *ot:cidadeMatriz,* a qual não é possível consultar, uma vez que esta informação não se encontra disponível em nenhuma das fontes.

(ii) Por outro lado, existem conceitos nas ontologias fonte que não estão mapeados para O_1, como por exemplo, as classes *a:PCHardware* e *e:Vendedor*. Isto sugere que, embora existentes nas fontes, estes conceitos não são de interesse da aplicação, pois não se encontram disponíveis para consulta na O_T.

(iii) Outro ponto que pode ser destacado diz respeito à ausência de regras de mapeamento entre algumas classes que aparentemente estão presentes em ambas as ontologias fonte e alvo. Por exemplo, as classes *v:Produto* e *a:Produto* não estão mapeadas. Isto ocorre porque, em O_1, não há informações sobre Equipamentos de Computador, o que implica que nem todos os tipos de produtos de O_1 correspondem a produtos de O_2. Com isso, são gerados mapeamentos apenas entre as subclasses que estão alinhadas.

3.3 Conclusões

Neste capítulo, discutimos como uma abordagem para Acesso a Dados baseado em Ontologias pode ser dividida em uma série de etapas, de forma que cada uma destas etapas possa ser tratada de maneira independente. Em seguida, descrevemos um ambiente, com seus respectivos módulos e atividades, através do qual este acesso pode ser desenvolvido. Dentro deste ambiente, caracterizamos os mapeamentos obtidos por meio do *módulo de gerador de mapeamentos entre as ontologias*, mostrando que tais mapeamentos podem ser gerados de maneira semi-automática. O próximo capítulo descreve, com detalhes, a nossa estratégia para a reescrita de consultas entre ontologias, a qual faz uso dos mapeamentos heterogêneos apresentados.

CAPÍTULO 4
Um Processo para Reescrita de Consultas entre Ontologias

Conforme apresentado no capítulo anterior, um dos serviços fundamentais em um ambiente de OBDA consiste em responder a uma consulta submetida sobre uma ontologia de domínio, considerando que a informação desejada pode estar armazenada em fontes de dados quaisquer. Para lidar com essa questão, propusemos trabalhar em uma estratégia na qual, inicialmente, nos abstraímos do tipo de fonte de dados e consideramos que tanto o esquema alvo quanto o esquema fonte são ontologias, facilitando a posterior tradução da consulta para o esquema da fonte. Com isso, faz-se necessário que enfrentemos o problema mais geral de reescrita de consultas entre duas ontologias, o qual será foco deste capítulo. A Seção 4.1 apresenta uma visão geral do processo de reescrita, enquanto a Seção 4.2 descreve a primeira etapa deste processo, que consiste na normalização da consulta recebida como entrada. A Seção 4.3 apresenta algumas noções de programação em lógica relevantes para o entendimento do restante do capítulo, uma vez que estas noções são utilizadas em atividades posteriores. A Seção 4.4 discute sobre a transformação de SPARQL para regras, descrevendo a geração da árvore que representa a consulta. A Seção 4.5 discorre sobre a reescrita da consulta, exibindo os algoritmos utilizados durante esta atividade. Finalmente, a Seção 4.6 exibe a geração da consulta local. Salientamos que não iremos detalhar conceitos sobre SPARQL. Estas informações podem ser encontradas no Apêndice A.

4.1 Visão Geral do Processo

O processo que propomos neste capítulo visa solucionar o problema de reescrita de consultas entre ontologias descrito a seguir. Seja uma ontologia alvo O_T, uma ontologia fonte O_S e um conjunto de mapeamentos entre ambas. Como reescrever uma consulta Q, submetida sobre O_T, em uma consulta equivalente sobre O_S, possibilitando a apresentação dos resultados de acordo com O_T e considerando que as ontologias são independentes e podem, dessa forma, possuir estruturas distintas? É importante destacar

que a solução aqui apresentada tem como uma de suas aplicações o cenário de *OBDA* descrito anteriormente. No entanto, ela é genérica o suficiente para ser utilizada em outros contextos onde haja a necessidade de reescrever consultas e reestruturar resultados entre ontologias.

O método que utilizamos para realizar o processo de reescrita é inspirado em noções de programação em lógica [Lloyd 1987]. A opção por esta abordagem foi motivada pela necessidade de manipularmos os mapeamentos heterogêneos expressos por meio de regras. Com isso, alguns dos conceitos já estabelecidos nesta área nos foram úteis, sendo realizadas as devidas adaptações, pois, ao contrário da maioria dos trabalhos que aplicam estas ideias, nós não utilizamos as regras para acessar diretamente os dados, mas sim para gerar as consultas reescritas. A Figura 4.1 apresenta uma visão geral do processo proposto, com as principais atividades ilustradas.

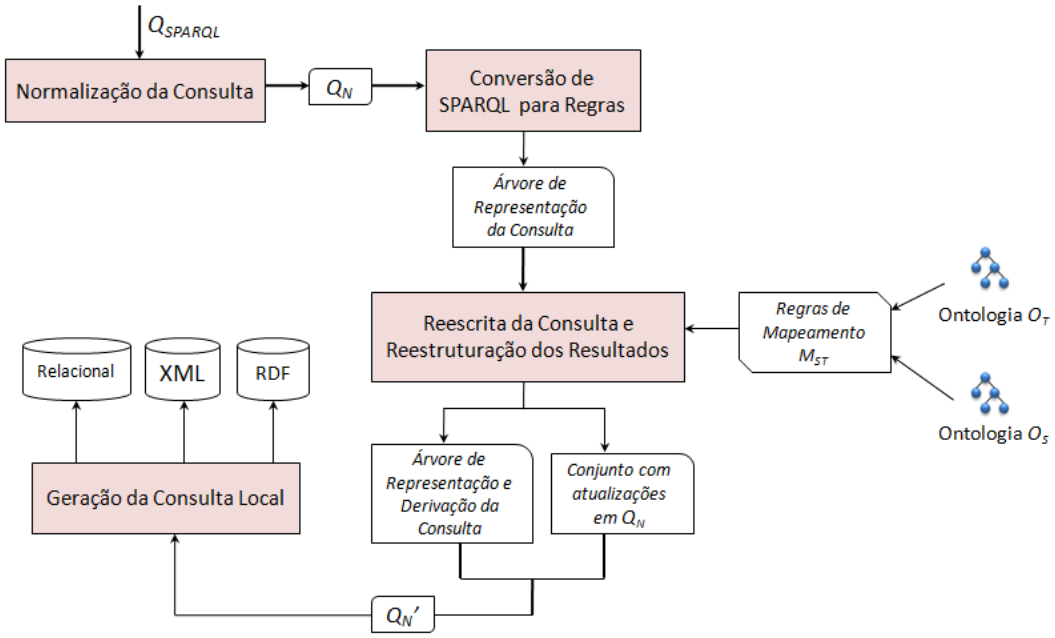


Figura 4.1. Visão Geral do Processo de Reescrita

De maneira geral, este processo recebe como entrada uma consulta SPARQL formulada em termos de O_T. A respeito desta consulta, fazemos as seguintes observações:

(i) Consideramos todos os construtores presentes na linguagem SPARQL, exceto GRAPH. Isto porque estamos tratando apenas de duas ontologias, de forma que a utilização do GRAPH torna-se irrelevante para a solução.

(ii) Focamos em consultas SELECT e CONSTRUCT, visto que estes são os dois formatos de retorno mais utilizados. Além disso, os demais formatos (ASK e

DESCRIBE) não trazem dificuldades adicionais em relação aos tratados, no que tange à reescrita.

A primeira atividade do processo de reescrita consiste na *Normalização da Consulta*, visando a geração de uma representação homogênea (Q_N) de Q_{SPARQL}. Em seguida, a *Conversão de SPARQL para Regras* cria uma árvore que refle a semântica, através de um conjunto de regras, do padrão de grafo de Q_N. Esta árvore, a qual chamamos de SPARQLD, é construída baseando-se em noções de programação em lógica, de forma que, durante a atividade de *Reescrita da Consulta* e *Reestruturação dos Resultados*, ela é complementada com as derivações obtidas utilizando as regras de mapeamento descritas no capítulo anterior (M_{ST}). A execução desta atividade produz dois artefatos como saída: a árvore SPARQLD com a consulta reescrita e um conjunto de atualizações em Q_N, as quais possibilitam que os resultados possam ser estruturados de acordo com a ontologia alvo, caso seja necessário. Finalmente, a partir destas saídas, é possível atualizar o conjunto Q_N para Q_N' e gerar uma consulta local que pode ser, de fato, executada sobre os dados. As seções subsequentes descrevem de maneira detalhada cada uma das atividades mencionadas.

4.2 Normalização da Consulta

Esta atividade tem como objetivo expressar em um formato homogêneo a consulta Q_{SPARQL} recebida como entrada, com o intuito de facilitar a manipulação desta consulta. Para isto, as tarefas I, II e III, descritas a seguir, são realizadas.

I. Extração dos blocos básicos de construção de uma consulta na linguagem SPARQL: formato de retorno, cláusula do *dataset*, padrão de grafo e modificadores de solução.

II. Criação de um CONSTRUCT padrão, se necessário: o retorno de uma consulta na forma de triplas da ontologia alvo é possível através da utilização da cláusula CONSTRUCT. Caso tal cláusula não tenha sido explicitamente construída, mas os resultados necessitem ser apresentados deste modo, um CONSTRUCT padrão será criado da maneira explanada a seguir. Seja R_S o conjunto de variáveis de retorno presentes na cláusula SELECT e P o padrão de grafo da consulta. Para cada variável v pertencente a R_S, serão buscados, em P, todos os padrões de triplas distintos nos quais v aparece.

III. Representação algébrica do padrão de grafo: a sintaxe oficial de SPARQL permite que padrões de triplas equivalentes sejam representados de maneiras distintas. Além disso, as expressões de padrões de grafos são construídas utilizando os operadores OPTIONAL, UNION, FILTER e o símbolo (.), que representa uma conjunção. Para agrupamento de padrões, esta sintaxe considera {} e algumas regras implícitas de precedência e associação. Por exemplo, o (.) tem precedência sobre o OPTIONAL, que por sua vez é associativo à esquerda [Pérez et al. 2009]. Diante disso, tendo em vista abandonar ambiguidades durante a análise da consulta nas atividades posteriores do processo de reescrita, nós representamos o padrão de grafo de Q_{SPARQL} seguindo o formalismo algébrico apresentado em [Arenas et al. 2009; Pérez et al. 2009]. Neste formalismo, são utilizados os operadores binários AND (.), UNION (UNION), OPT (OPTIONAL) e FILTER (FILTER). Como exemplificação, a Figura 4.3 exibe a forma algébrica do trecho de consulta da Figura 4.2.

```
{$x <http://xmlns.com/foaf/idade> $y ;
    <http://xmlns.com/foaf/conhece> $z .
 ?z f:pais ?c .
 FILTER (?y > 30 && ?y < 50).
 OPTIONAL {?z f:fone ?p}
 FILTER ($c = "Chile")}
```

Figura 4.2. Trecho de uma consulta SPARQL com sintaxe mista

```
  ( (   (?x, http://xmlns.com/foaf/idade, ?y )
    AND (?x, http://xmlns.com/foaf/conhece, ?z)
    AND (?z, http://xmlns.com/foaf/pais, ?c))
  OPT (?z, http://xmlns.com/foaf/fone, ?p))
FILTER (((?y > 30) && (?y < 50))∧ ?c = "Chile"))
```

Figura 4.3. Representação algébrica da consulta apresentada na Figura 4.2

Observe que a consulta da Figura 4.2 possui uma sintaxe mista: as variáveis estão simbolizadas tanto com "?" quanto com "$"; alguns predicados dos padrões de triplas estão retratados com a URI completa, enquanto outros estão apenas com os prefixos; as conjunções entre os três primeiros padrões de triplas estão expressas de formas distintas. Por outro lado, na representação algébrica da Figura 4.3, todas estas diferenças são eliminadas. Além disso, os filtros são agrupados, uma vez que possuem o mesmo escopo de aplicação e as expressões parentizadas, tornando explícita a precedência e a associação dos operadores. Esta parentização é baseada tanto na semântica da linguagem quanto na estruturação estabelecida na consulta de entrada.

Após a realização das tarefas que caracterizam a atividade de Normalização, é gerada uma consulta Q_N, conforme definido abaixo.

Definição 4.1. Uma consulta normalizada Q_N é uma quádrupla Q_N = (*R, P, DS, SM*), onde:

- *R* é o formato de retorno da consulta. Subdividimos este conjunto nas seguintes partes, onde uma destas partes, mas não ambas, pode ser vazia:
  - R_S: Conjunto de variáveis do retorno, caso a consulta seja uma seleção.
  - R_C: Padrão de grafo do CONSTRUCT, o qual pode ser informado pelo usuário ou construído da forma padrão descrita anteriormente.
- *P* é o padrão de *matching* do grafo, expresso em uma notação algébrica.
- *DS* é a ontologia sobre a qual a consulta é submetida, neste caso, O_T.
- *SM* é o conjunto de modificadores de solução da consulta.

Visando ilustrar a entrada e a saída desta atividade, apresentamos o Exemplo 4.1. Por questões de legibilidade, as URIs encontram-se representadas por seus respectivos prefixos.

Exemplo 4.1. Seja a seguinte consulta sobre a ontologia de *Vendas* O_1(exibida no capítulo anterior): "*Recupere o título e o autor de todos os livros que custam mais de 30 reais. Caso as informações sobre a editora vinculada ao livro estejam disponíveis, recupere tal editora, juntamente com o nome e o endereço da mesma. Além disso, retorne o título de todos os Vídeos, com seus respectivos diretores. Ordene os resultados pelo título.*"

Entrada: Q_{SPARQL} =

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
PREFIX ven: <http://vendas.com#>

SELECT ?tl ?au ?e ?ne ?ee ?tv ?dir
FROM <Vendas.owl>
WHERE
{
   {   ?l ven:titulo ?tl .
       ?l ven:tipo ?k .
       ?l ven:autor ?au
       OPTIONAL
       {
         ?e ven:nomeEditora ?ne .
         ?e ven:endereco ?ee.
         ?l ven:publicadaPor ?e

        }
       ?l ven:preco ?p .
       FILTER (?k = 'livro ' && ?p > 30)
   }
   UNION
   {
       ?v ven:titulo ?tv .
       ?v rdf:type ven:Video .
       ?v ven:diretor ?dir
   }
} ORDER BY ?tl ?tv
```

Saída: Q_N =

R_S =	{?tl,?au,?e,?ne,?ee,?tv,?dir}
R_C =	{ (?l ven:titulo ?tl) (?l ven:autor ?au) (?l ven:publicadaPor ?e) (?e ven:nomeEditora ?ne) (?v ven:titulo ?tv) (?v ven:diretor ?dir) }

P =	{
	((((((?l, ven:titulo, ?ti)
	AND (?l, ven:tipo, k)
	AND (?l, ven:autor, au))
	OPT ((?e, ven:nomeEditora, ?ne)
	AND (?e, ven:endereco, ?ne)
	AND (?l, ven:publicadaPor, ?e))
	AND (?l, ven:preco, ?p))
	FILTER ((?k = "livro") && (?p > 30)))
	UNION (
	(?v, ven:titulo, ?tv)
	AND (?v, rdf:type, ven:Video)
	AND (?v, ven:director, ?dir))
	}

DS =	{Vendas.owl}
SM =	{(ORDER,<?ti,?tv>)}

4.3 Algumas Noções de Programação em Lógica

Antes de prosseguirmos com a descrição da estratégia de reescrita, iremos abordar algumas noções de programação em lógica [Lloyd 1987; Lloyd and Shepherdson 1991] relevantes para a explanação do restante do nosso processo.

Definição 4.2. Uma *cláusula de um programa definitivo* é uma cláusula da forma

$$A \leftarrow B_1,..., B_n$$

onde A é um átomo e $B_1,..., B_n$ é uma conjunção de átomos. O lado esquerdo da cláusula é chamado de cabeça e o lado direito, de corpo. Tanto o corpo quanto a cabeça de uma cláusula podem ser vazios. Quando o corpo é vazio, a cláusula é chamada de fato, sendo escrita apenas como A. Se a cabeça for vazia, a cláusula é chamada de *objetivo definitivo*, sendo escrita da seguinte maneira: $\leftarrow B_1,..., B_n$. Uma cláusula pode ainda ser chamada de regra.

Definição 4.3. Um *programa definitivo* é um conjunto finito de *cláusulas de programa definitivo*.

As cláusulas, em programação em lógica, são utilizadas para derivar fórmulas a partir de outras fórmulas. Para esta derivação, é necessário utilizar a ideia de unificação, apresentada através das definições a seguir.

Definição 4.4. Uma *substituição* Θ é um conjunto finito da forma $\{v_1/t_1,..., v_n/t_n\}$, onde: cada v_i é uma variável, cada t_i é um termo (variável, constante ou função) diferente de v_i e as variáveis $v_1,...,v_n$ são distintas. Neste conjunto, cada elemento v_i/t_i é chamado de *ligação* (*binding*) para v_i. Seja E uma expressão. EΘ, chamada de *instância de E por* Θ, é uma nova expressão obtida substituindo, simultaneamente, cada ocorrência da variável v_i em E pelo termo t_i.

Definição 4.5. Um *unificador* de duas expressões é uma substituição que torna tais expressões iguais, desde que isto seja possível. Um unificador mais geral (*mgu*: *most general unifier*) é um unificador que possui o número mínimo de substituições.

No Apêndice B, é possível visualizar o algoritmo de unificação utilizado nesta dissertação. Em suma, tal algoritmo recebe como entrada duas expressões e retorna o *mgu*, caso ele exista, destas expressões. De posse das definições de unificação, podemos formalizar a noção de derivação conforme descrito abaixo.

Definição 4.7. Seja G um objetivo da forma $\leftarrow A_1,..., A_m,..., A_k$ e C uma cláusula do tipo $A \leftarrow A_1,..., B_1,..., B_q$. Então, G' é *derivado* a partir de G e C, utilizando o *mgu* Θ, se as seguintes condições forem satisfeitas:

(i) A_m é um átomo, chamado de átomo *selecionado*, em G.

(ii) é um *mgu* de A_m e A.

(iii) G' é o objetivo $(A_1,..., A_{m-1}, B_1,..., B_q, A_{m+1}, ..., A_k)$.

As possíveis derivações de um objetivo, usando um programa, são representadas por uma árvore especial, chamada de árvore SLD, descrita a seguir.

Definição 4.8. Seja P um programa e G um objetivo. Uma árvore SLD para P $\cup$ {G} é uma árvore que satisfaz as seguintes condições:

(i) Cada nó da árvore é um objetivo (possivelmente vazio).

(ii) A raiz é o nó G.

(iii) Seja $A_1,..., A_m,..., A_k$ (k 1) um nó pertencente à árvore e suponha que A_m é o átomo selecionado. Então, para cada cláusula de entrada do tipo A $B_1,...,$ B_q, tal que A_m e A são unificáveis com o *mgu* , o nó tem um filho

$$(A_1,..., A_{m-1}, B_1 ,..., B_{q,} A_{m+1}, ..., A_k) \quad .$$

(iv) Nós que são cláusulas vazias não possuem filhos.

Cada ramo de uma árvore SLD é uma derivação de P U {G}. Os ramos correspondentes a derivações bem sucedidas, infinitas e falhas são chamados, respectivamente, de *ramos bem sucedidos*, *ramos infinitos* e *ramos falhos*. Diz-se que um ramo é falho se ele termina em um nó tal que o átomo selecionado não unifica com a cabeça de nenhuma das cláusulas do programa. Neste trabalho, nós utilizamos as características fundamentais de construção da SLD, porém com determinadas modificações, conforme discutiremos mais adiante.

4.4 Conversão de SPARQL para Regras

Nesta seção, descrevemos como uma consulta SPARQL pode ser representada através de um conjunto de regras. Além disso, apresentamos a construção de uma árvore, baseada na SLD, que reflete tal conjunto e, consequentemente, a consulta recebida como entrada. É válido salientar que utilizamos esta árvore pelos seguintes motivos: (i) a SPARQLD retrata as regras em um formato adequado para as manipulações e operações que realizamos tanto durante a atividade de *Reescrita da Consulta e Reestruturação dos Resultados* quanto para a *Geração da Consulta Local*; (ii) por tratar-se de uma estrutura mais complexa, ela permite a utilização de anotações úteis em ambas as atividades mencionadas.

Destacamos ainda que a SPARQLD de uma consulta Q_{SPARQL} descreve tal consulta realizando, possivelmente, algumas modificações nesta, como por exemplo: a aplicação da distributividade para os operadores AND e UNION; o agrupamento das operações de junção. Estas modificações, as quais são consequência da transformação de Q_{SPARQL} para as regras, assim como da construção da SPARQLD, mantêm a semântica da consulta original e possibilitam a realização de alguns dos procedimentos desempenhados na atividade de *Reescrita da Consulta e Reestruturação dos Resultados*.

4.4.1 Considerações Iniciais

A tradução de SPARQL para regras (mais especificamente, Datalog) foi anteriormente abordada em alguns trabalhos existentes [Schenk 2007; Polleres 2007; Ianni et. al 2009]. Nestes trabalhos, são apresentadas discussões a respeito da semântica formal da linguagem SPARQL, baseando-se em resultados já consolidados no campo da lógica. Além disso, tais resultados são reaproveitados para realização de análises, formalizações e extensões em torno desta linguagem.

Neste contexto, as abordagens mais relevantes foram apresentadas por Polleres e Ianni [Polleres 2007; Ianni et. al 2009]. Em suma, a ideia chave destes autores é propor um RDF *Store*, nomeado de *GiaBATA* [Ianni et. al 2009], no qual o processamento das consultas é realizado no topo de um motor de inferência Datalog. Para tanto, duas atividades são executadas: (i) os dados RDF, ao serem armazenados no repositório, são transformados em um conjunto de fatos, constituindo, assim, a base de conhecimento de um programa Datalog-ASP (*Aswer Set Programming*); (ii) As consultas SPARQL recebidas como entrada são mapeadas para um conjunto de regras, sendo consideradas as devidas exceções para os elementos que não podem ser reproduzidos diretamente em Datalog. Como exemplo de tais elementos, podemos citar alguns construtores presentes no FILTRO (*isBlank*, *isIRI*, *isLiteral*, *Bound*, etc). Para que estes construtores pudessem ser utilizados, foram definidos predicados externos que os representassem, de forma que, ao longo do processamento da consulta, são realizadas chamadas a estes predicados.

Como resultado desta ideia, os autores mostraram que a reprodução da semântica de SPARQL através de regras pode ser um bom artifício para o desenvolvimento de estudos formais sobre esta linguagem. Além disso, em algumas situações, o RDF *Store* proposto apresentou um desempenho melhor que os atualmente existentes.

É importante salientar que, embora a transformação apresentada por Polleres e Ianni tenha servido como ponto inicial para nossa representação, uma série de fatores nos distingue deste trabalho. Primeiro, porque o nosso foco é a reescrita e não o processamento de consultas em um RDF *Store*. Isto significa que nós não utilizamos as regras para acessar diretamente os dados, pois não temos a intenção de exportar tais dados para uma base de conhecimento Datalog. O nosso intuito é combinar as regras

oriundas da consulta de entrada com as regras de mapeamento para gerar as consultas reescritas.

Dada esta diferença fundamental, nós pudemos utilizar meios distintos para expressar construtores mais complexos de SPARQL, tais como o OPT e as restrições de filtro. No caso do OPT, por exemplo, a semântica de tal operador indica que todo o fragmento de consulta sob seu escopo pode, ou não, contribuir com algum resultado. Em [Polleres 2007; Ianni et. al 2009], esta semântica é representada através da negação por falha, ou seja, a operação A *OPT* B é equivalente a: (A AND B) OR (A AND *not* B). Por outro lado, construtores de filtro são expressos por meio de predicados externos. Em nossa abordagem, isto não é necessário, ao contrário, traria apenas gastos adicionais, uma vez que as regras geradas não serão avaliadas diretamente contra os fatos. Por este motivo, além das regras, optamos pela utilização de anotações na árvore SPARQLD, visando retratar as informações relevantes para a reescrita da consulta.

4.4.2 Geração da Árvore SPARQLD

Conforme definido anteriormente, uma árvore SLD é uma árvore especial que tem como finalidade representar as possíveis derivações de um objetivo G, vinculado a um programa P. Em sistemas PROLOG, esta árvore representa o espaço de busca e resolução que reflete a pilha de objetivos construída durante a computação de G. Como resultado desta computação, é fornecida uma resposta que, em geral, consiste em uma afirmação (ou negação) ou ainda nas instâncias que satisfazem G. Neste trabalho, nós não estamos interessados em receber uma resposta, mas sim em gerar um novo conjunto de regras que representem a consulta reescrita. Dessa forma, temos que G corresponde ao questionamento da consulta, isto é, às informações solicitadas na cláusula SELECT (ou CONSTRUCT), as quais são armazenadas no conjunto R de Q_N (consulta normalizada). Já o programa P é constituído pelas regras referentes à consulta, juntamente com as regras de mapeamento.

Uma árvore SPARQLD é construída satisfazendo as condições de uma SLD. No entanto, para que seja possível representar adequadamente a semântica dos construtores de SPARQL, bem como informações necessárias durante o processo de reescrita, os nós desta árvore são enriquecidos com anotações, que são caracterizadas a seguir.

Definição 4.9. Uma *árvore SPARQLD* é uma árvore, com construção inspirada na SLD, que reflete uma consulta SPARQL. Além disso, cada um de seus nós possui as seguintes características:

- Um identificador de referência (*idRef*) delimitando a subárvore que reflete a semântica do padrão de grafo correspondente ao nó. Na prática, isto indica qual subárvore deve ser descartada no caso de falha de um determinado nó. Por exemplo, se um nó com *idRef =2* for um nó falho, toda a subárvore, partindo da raiz, que possuir o *idRef* com este valor deve ser excluída da consulta.

- Uma lista de predicados, obtidos tanto a partir da consulta de entrada quanto a partir das regras de mapeamento, de forma que cada um destes predicados é sinalizado como sendo, ou não, uma informação de contexto. Os predicados obtidos diretamente da consulta nunca representarão contexto. Entretanto, ao longo do processo de reescrita, aqueles adquiridos a partir dos mapeamentos podem expressar tal informação. É importante ressaltar que os predicados mencionados neste trabalho podem ser classificados conforme a Definição 4.10.

Definição 4.10. Seja $A = p_i(t_1,..., t_n)$ um predicado, onde p_i é um símbolo de predicado e t_n são os seus argumentos, os quais são termos. Dizemos que *A* é um *predicado referente a um padrão de grafo*, que pode ou não ser um padrão de tripla, se ele representa um padrão de grafo qualquer da consulta. Neste caso, o expressamos com a notação q_i, isto é, o símbolo de predicado *p* possui o valor *q*. Por outro lado, *A* é um *predicado referente a um conceito da ontologia* se ele sempre representa um padrão de tripla RDF que reflete uma classe ou uma propriedade. Neste caso, *A* pode ser unário (classe) ou binário (propriedade) e o símbolo de predicado *p* é igual à URI da classe ou da propriedade que ele retrata.

- Uma indicação denotando o operador que compõe o padrão de grafo (*GP*) referente a um nó ou aos filhos deste nó.

- Uma referência para os identificadores dos filtros que são aplicados sobre o nó. Isto porque cada expressão de filtro contida na consulta é simbolizada através de uma árvore sintática, visando facilitar tanto a busca quanto a manipulação nestas expressões. Uma vez que cada uma destas árvores de filtro é rotulada com um identificador, os nós da SPARQLD armazenam estes identificadores, dividindo-os em dois grupos: (i) filtro aplicado diretamente sobre o padrão correspondente ao nó (*filter*); (ii) filtros aplicados a outros padrões mais externos, nos quais o nó esteja inserido (*filtersExt*). Esta divisão é necessária para que o escopo de aplicação dos filtros possa ser mantido durante a geração da consulta reescrita.

Como ilustração, a Figura 4.4 exibe um exemplo de nó de uma árvore

SPARQLD (árvore da Figura 4.8). Nele, temos que: (i) o identificador de referência (*idRef*) é igual a 2; (ii) o padrão de grafo retratado é um padrão com a operação UNION, que é realizada entre os filhos deste nó; (iii) o filtro (*filter*) com identificador igual a F_4 é aplicado diretamente sobre o padrão; (iv) não há filtros externos (*filtersExt*); (v) a lista de predicados do nó é igual {*ven:titulo(l,tl)*, *ven:tipo(l,k)*, *ven:autor(l,au)*, *ven:preco(l,p)*}. Observe que todos estes são *predicados referentes a um conceito da ontologia*.

idRef = 2	filter = {F_4}
GP = UNION	filtersExt = { }
ven:titulo(l,tl),ven:tipo(l,k), ven:autor(l,au), ven:preco(l,p)	

Figura 4.4. Exemplo de um nó de uma árvore SPARQLD

Além destas características apresentadas, uma árvore SPARQLD difere da SLD quanto às condições de parada. No nosso caso, o processo termina quando a consulta de entrada encontra-se completamente reescrita em termos da ontologia fonte. Assim, não temos derivações (ramos) *infinitas*, uma vez que as regras de mapeamento são finitas e não recursivas. Um ramo é *falho* se ele gera uma subconsulta redundante, de retorno vazio ou que não corresponda ao questionamento da consulta original. Por estes motivos, ramos *falhos* não são incorporados na consulta. Caso contrário, o ramo é dito *bem sucedido* e está apto a compor a consulta reescrita. Os algoritmos apresentados na próxima etapa (reescrita) lidam com estas questões.

A seguir, descrevemos, inicialmente, o algoritmo para geração de um conjunto de regras a partir de uma consulta SPARQL. Em seguida, de posse destas regras, explanamos o algoritmo que constrói a árvore de representação da consulta.

Geração de um Conjunto de Regras a partir de uma consulta SPARQL (Algoritmo *GeraRegraSPARQL*)

A Figura 4.5 exibe o algoritmo *GeraRegraSPARQL*, o qual é responsável pela geração das regras, enquanto a Figura 4.5 ilustra a saída deste algoritmo para a consulta apresentada no Exemplo 4.1. Tal algoritmo recebe como entrada um conjunto de variáveis (*V*) relativas a um padrão de grafo, o próprio padrão de grafo (*GP*) e uma variável *i* que controla a geração das regras na recursão. Na primeira chamada, estes parâmetros de entrada recebem os seguintes valores: *V* = variáveis do padrão *P* da consulta normalizada Q_N; *GP* = padrão *P* de Q_N; *i* =1. Para o exemplo da Figura 4.6,

estas atribuições são (observe a saída Q_N do Exemplo 4.1): V = *vars*(P) = {*au,dir,e,ee,k,l,ne,p,tl,tv,v*}; GP = o conteúdo de P; i = 1.

ENTRADA: Um conjunto de variáveis de um padrão de grafo, o padrão de grafo e um número inteiro.
SAÍDA: Um conjunto de regras que representa a consulta relativa ao padrão de entrada.

GeraRegraSPARQL(V, GP, i)

1. **No caso**
2. **Caso 1: Se** GP = TB (*Triple Block*) **Então**
3. q_i(V) ← predicados do TB;
4. **Caso 2: Se** GP = P' FILTER R **Então**
5. q_i(V) ← q_{2*i}(vars(P')),F_{2*i} (R)
6. *geraRegraSPARQL(vars(P'), P', 2*i);*
7. **Caso 3: Se** GP = P' AND P'' **Então**
8. q_i(V) ← q_{2*i}(vars(P')), q_{2*i+1}(vars(P''));
9. *geraRegraSPARQL(vars(P'), P', 2*i);*
10. *geraRegraSPARQL(vars(P''), P'', 2*i+1);*
11. **Caso 4: Se** GP = P' UNION P'' **Então**
12. q_i(V[vars(P'')=null]) ← q_{2*i}(vars(P'));
13. q_i(V[vars(P')=null]) ← q_{2*i+1}(vars(P''));
14. *geraRegraSPARQL(vars(P'), P', 2*i);*
15. *geraRegraSPARQL(vars(P''), P'', 2*i+1);*
16. **Caso 5: Se** GP = P' OPT P'' **Então**
17. q_i(V) ← q_{2*i}(vars(P'));
18. q_i(V) ← q_{2*i}(vars(P')), *opt(true),* q_{2*i+1}(vars(P''));
19. *geraRegraSPARQL(vars(P'), P', 2*i);*
20. *geraRegraSPARQL(vars(P''), P'', 2*i+1);*
22. **Fim caso**

Fim ***GeraRegraSPARQL***

Figura 4.5. Algoritmo *GeraRegraSPARQL*

Durante o funcionamento do *GeraRegraSPARQL*, o padrão P recebido como entrada vai sendo analisado de maneira recursiva, do operador mais externo para os mais internos. Em cada chamada, são geradas as regras relativas ao padrão vinculado ao operador examinado, bem como são realizadas novas chamas recursivas para os subpadrões de P. Estas atividades são desempenhadas de acordo com os casos descritos a seguir. Em todos estes casos, a cabeça da regra é o *predicado referente ao padrão de grafo* que chamou a recursão. Por exemplo, $q_1(au,dir,e,ee,k,l,ne,p,tl,tv,v)$ é o predicado referente ao padrão de grafo mais externo da consulta do Exemplo 4.1.

Regras da Consulta:

```
1.  q1(au,dir,e,ee,k,l,ne,p,tl,tv,v)  ←  q2(au,e,ee,k,l,ne,p,tl) // dir,tv,v = null

2.  q1(au,dir,e,ee,k,l,ne,p,tl,tv,v)  ←  q3(dir,tv,v) //  au,e,ee,k,l,ne,p,tl = null

3.          q2(au,e,ee,k,l,ne,p,tl)   ←  q4(au,e,ee,k,l,ne,p,tl),
                                         F4(((?k = "livro") && (?p > 30)))

4.          q4(au,e,ee,k,l,ne,p,tl)   ←  q8(au,e,ee,k,l,ne,tl),q9(l,p)

5.            q8(au,e,ee,k,l,ne,tl)   ←  q16(au,k,l,tl)

6.            q8(au,e,ee,k,l,ne,tl)   ←  q16(au,k,l,tl),opt(true),q17(e,ee,l,ne)

7.                    q16(au,k,l,tl)  ←  ven:titulo(l,tl),ven:tipo(l,k),
                                         ven:autor(l,au)

8.                    q17(e,ee,l,ne)  ←  ven:nomeEditora(e,ne),ven:endereco(e,ee),
                                         ven:publicadaPor(l,e)

9.                          q9(l,p)   ←  ven:preco(l,p)

10.                   q3(dir,tv,v)    ←  ven:titulo(v,tv),ven:Video(v),
                                         ven:diretor(v,dir)
```

Figura 4.6. Conjunto de regras correspondentes à consulta do Exemplo 4.1

• Caso 1: consiste no caso base do algoritmo, correspondente a um conjunto de padrões de triplas relacionadas apenas através do operador AND. Tais padrões não possuem, portanto, outros padrões aninhados. Nesta situação, é criada uma única regra, cujo corpo é composto pela conjunção dos predicados que retratam estes padrões (linha 3). Neste trabalho, os conceitos da ontologia presentes na consulta são expressos através de predicados unários e binários (veja Definição 4.10), os quais são obtidos a partir dos padrões de triplas *(s, p,o)* da consulta SPARQL, da seguinte forma:

  - Um padrão de tripla do tipo (?x||A) rdf:type uriBase:C, onde o sujeito é uma variável *x* ou uma *URI A*, é escrito como um predicado unário da forma uri:C(x||A). Caso haja uma variável na posição da classe *C*, isto é, o padrão de tripla seja da forma (?x||A) rdf:type ?y, a classe correspondente a esta variável deve ser recuperada no FILTRO e utilizada para construção do predicado.
  - Um padrão de tripla do tipo (?x||A)uriBase:pred (?y||B||'b'), onde o sujeito é uma variável *x* ou uma *URI A* e o objeto é uma variável *y*, uma *URI B* ou uma constante *b*, é representado como um predicado binário da forma uri:pred(x||A,y||B||'b').

• Caso 2: origina uma regra para o operador FILTER (linha 5), assim como realiza uma chamada recursiva para produzir as regras do padrão P', sobre o qual a restrição está sendo aplicada.

• Caso 3: produz uma única regra (linha 8), onde o corpo é uma conjunção entre os predicados que representam aos padrões de grafo P' e P'', os quais estão relacionados através do operador AND. Além disso, são realizadas duas chamadas recursivas visando gerar as regras referentes à P' e P''.

• Caso 4: gera duas regras (linhas 12 e13) representando uma disjunção entre os predicados relativos aos padrões P' e P'', os quais estão associados através do operador UNION. Assim como no caso anterior, são efetuadas chamadas recursivas para os subpadrões P' e P''.

• Caso 5: origina duas regras que caracterizam a operação P' OPT P'', isto é, a presença ou não de P''. Para isto, na primeira regra (linha 17), há apenas o predicado correspondente a P', representando a negação por falha de P'', ou seja, a ausência deste padrão. Já a segunda regra (linha 18) reflete a existência de ambos P' e P'', com um predicado adicional denotando a marcação do padrão que é opcional. As chamadas recursivas ocorrem de maneira semelhante aos casos três e quatro.

Com o intuito de ilustrar a execução do *GeraRegraSPARQL*, considere as regras exibidas na Figura 4.6. Neste exemplo, o algoritmo entra primeiramente no Caso 4, pois o operador mais externo de P é o UNION. A partir de então, são geradas as regras 1 e 2. Em seguida, são realizadas chamadas recursivas para os padrões P' e P'' envolvidos na união. A partir de P', a seguinte sequencia de padrões é executada: *P' FILTER R*, *P' AND P''* e *P' OPT P''*, o que leva a geração das regras 2 a 8. Já P'' corresponde ao caso base, originando, portanto, apenas a regra 9.

Geração da Árvore SPARQLD que Reflete um Conjunto de Regras (Algoritmo *ConstroiSPARQLD*)

Uma vez obtido o conjunto de regras relativas à Q_N, procedemos com a construção da árvore SPARQLD, através do algoritmo C*onstroiSPARQLD*, exibido na Figura 4.7. Já a Figura 4.8 apresenta a árvore SPARQLD correspondente às regras da Figura 4.6. Observe que esta árvore reflete as derivações efetuadas utilizando estas regras. Note ainda que, por questões de praticidade e legibilidade, novos nós são criados apenas durante a derivação de predicados que originam mais de uma possibilidade de caminho. Tais predicados representam padrões de grafos não básicos (*Group Graph Pattern* – GGP), ou seja, padrões com os operadores UNION e OPT. A seguir, listamos as tarefas desempenhadas pelo *ConstroiSPARQLD*.

Funcionamento geral do *ConstroiSPARQLD*

1. Cria a raiz da árvore (linhas 1-3)

A raiz da árvore possui o predicado referente ao padrão de grafo com o operador mais externo da consulta.

2. Invoca o procedimento *constroiDerivacoes*, passando a raiz, com o objetivo de montar o restante da árvore (linha 4).

3. Retorna o nó raiz da árvore, que agora é a própria árvore SPARQLD de representação da consulta (linha 5)

Funcionamento do procedimento *constroiDerivacoes*

Neste procedimento, *p* é o predicado que está sendo analisado, o qual é sempre o primeiro da lista de predicados do nó atual (linha 5).

1. Se *p* representa um filtro, gera a árvore (rotulada de F_K) e as referências deste filtro. Em seguida, remove *p* dos predicados do nó, pois, uma vez informadas as referências do filtro, tal predicado não é mais necessário (linhas 7-10).

ENTRADA: Conjunto R das regras que representam um padrão de grafo P.
SAÍDA: Árvore SPARQLD que representa tal padrão.

ConstroiSPARQLD(R)

```
1.  Criar novo nó T; //nó raiz da SPARQLD
2.  T. predicados := q1(vars(P));
3.  T.idRef := 1;
4.  constroiDerivacoes(T);
5.  Retorne T;
```

Fim ConstroiSPARQLD.

constroiDerivacoes(noAtual)

```
6.  Seja p o predicado atual analisado em noAtual.predicados;
7.  Se p = FK(R) Então  //restrição de filtro
8.    FK := gerarArvoreFiltro(R);
9.    Adicionar FK como filtro (filter) para noAtual;
10.   Remover FK dos predicados de noAtual;
11.   constroiDerivacoes(noAtual);
12. Senão
13.   Sabemos que R é constituído por um conjunto de regras do tipo H ← B, onde H é um átomo e B,
      uma conjunção de átomos.
14.   Seja u o número de unificações de p com as cabeças H das regras contidas em R.
15.   No Caso
16.     Caso 1: Se u = 0 Então
17.       noAtual.GP := BGP;
18.     Caso 2: Se u = 1 Então
19.       Seja p = qi e B o corpo da regra com a qual qi unifica com H.
20.       noAtual.predicados := predicados[qi/B];
21.       constroiDerivacoes(noAtual);
22.     Caso 3: Se u = 2 Então
23.       Criar os nós no1 e no2 para P' e P'', respectivamente;
24.       Seja p = qi.
25.       Seja B1 e B2 o corpo das regras x e y, respectivamente, com as quais qi unifica com H, onde x < y.
26.       Se B2 possui o predicado opt(true) Então
27.         Por construção, sabemos que B1 = q2*i e B2 = q2*i, opt(true), q2*i+1
28.         noAtual.GP = OPTIONAL;
29.         no1.predicados := predicados[qi/B1];
30.         no2.predicados := q2*i+1;
31.         no1.idRef := noAtual.idRef;
32.         no2.idRef := 2*i+1;
33.         Preencher as indicações de filtros de no1 com as mesmas de noAtual;
34.         Adicionar todos os filtros (filter e filtersExt) de noAtual nos filtros externos (filtersExt) de no2;
35.       Senão
36.         noAtual.GP = UNION;
37.         no1.predicados := predicados[qi/B1];
38.         no2.predicados := predicados[qi/B2];
39.         no1.idRef := 2* i;
40.         no2.idRef = 2*i+1;
41.         Adicionar todos os filtros (filter e filtersExt) de noAtual nos filtros externos (filtersExt) de no1 e no2;
42.       Fim Se
43.       Adicionar no1 e no2 como filhos de noAtual;
44.       constroiDerivacoes(no1);
45.       constroiDerivacoes(no2);
46.   Fim Caso
47. Fim Se
```

Fim constroiDerivacoes

Figura 4.7. Algoritmo *constroiSPARQLD*

2. Se *p* não é filtro, verifica as regras com as quais *p* unifica (linhas 12-14).

3. Se não há unificação (Caso 1), apenas armazena a informação de que o nó retrata um BGP. É o caso base do algoritmo (linhas 16 e 17).

Podemos concluir que é um BGP porque a ausência de unificações indica que todos os predicados existentes no *noAtual* são *predicados referente a um conceito da ontologia*, ou seja, um padrão de tripla ou uma conjunção destes padrões.

4. Se há apenas uma unificação (Caso 2), substitui *p* pelo corpo da regra (linha 20).

Nesta situação, não há necessidade de criar novos nós, mas somente de substituir q_i pelo corpo *B* da regra H ← B com a qual q_i unificou com *H*. A notação *predicados*$[q_i/B]$ indica que q_i está sendo substituído por *B* nos *predicados* contidos no *noAtual*. Na prática, q_i é removido do início da lista e *B* inserido no final, de forma que a operação de substituição sempre ocorre em tempo constante e os predicados que ainda precisam ser analisados permanecem nas primeiras posições da lista.

5. Se há duas unificações, cria dois novos nós e verifica se *p* representa um padrão de grafo com UNION ou OPT (linhas 22-26).

A existência de mais de uma unificação aponta dois possíveis caminhos na derivação. Isto sugere o operador UNION ou o OPT.

6. Para ambos OPT e UNION, o nó atual é anotado com o respectivo operador (linhas 28 e 36). Os predicados e os filtros dos novos nós são preenchidos de acordo com a operação verificada (linhas 29-34 e 37- 41). Além disso, estes nós criados são adicionados como filhos do nó atual (linha 43).

Mais especificamente, no OPT, o *no1* recebe os predicados do *noAtual*, substituindo q_i por *B1* (linha 29). Além disso, o *idRef* do *no1* permanece o mesmo do *noAtual* (linha 31), assim como as referências dos filtros (linha 33). Por outro lado, o *no2* recebe: (i) somente a parte opcional de *B2* (linha 30); (ii) um novo valor de identificação (linha 32); (ii) uma anotação sinalizando que os filtros do *noAtual* pertencem ao conjunto de filtros externos (linha 34) do *no2*. Esta diferença na criação dos nós para o OPT ocorre pelo seguinte motivo: o OPT é uma conjunção na qual um trecho desta não precisa ser satisfeita. Dessa forma, a parte não opcional (*no1*) é tratada com a mesma semântica do AND. Por outro lado, o trecho opcional é isolado pelo algoritmo, que não mantém os demais predicados do *noAtual* no *no2*, bem como cria novos identificadores de referência para este nó. Em termos práticos, a ideia é possibilitar que o trecho opcional seja analisado sozinho durante o algoritmo de reescrita, de maneira a não interferir negativamente no restante da consulta. Por exemplo, se na Figura 4.8 algum nó de *idRef = 2* for falho, toda subárvore abaixo do primeiro nó com este identificador deve ser excluída, incluindo o *idRef = 17*. No

entanto, se somente a parte opcional (*idRef = 17*) for falha, apenas a subárvore a partir de *idRef = 17* é descartada da consulta.

No UNION, os dois novos nós *no1* e *no2* recebem os predicados do *noAtual*, substituindo q_i por *B1* e *B2*, respectivamente (linhas 37 e 38). Os *idRefs* recebem novos valores (linhas 39 e 40) e as referências de filtro do *noAtual* são indicados como filtros externos de *no1* e *no2* (linha 41).

Finalmente, como pode ser observado neste algoritmo, novas chamadas recursivas são sempre realizadas, tendo em vista examinar os predicados restantes, até que se chegue ao Caso 1 (o nó retrata um BGP).

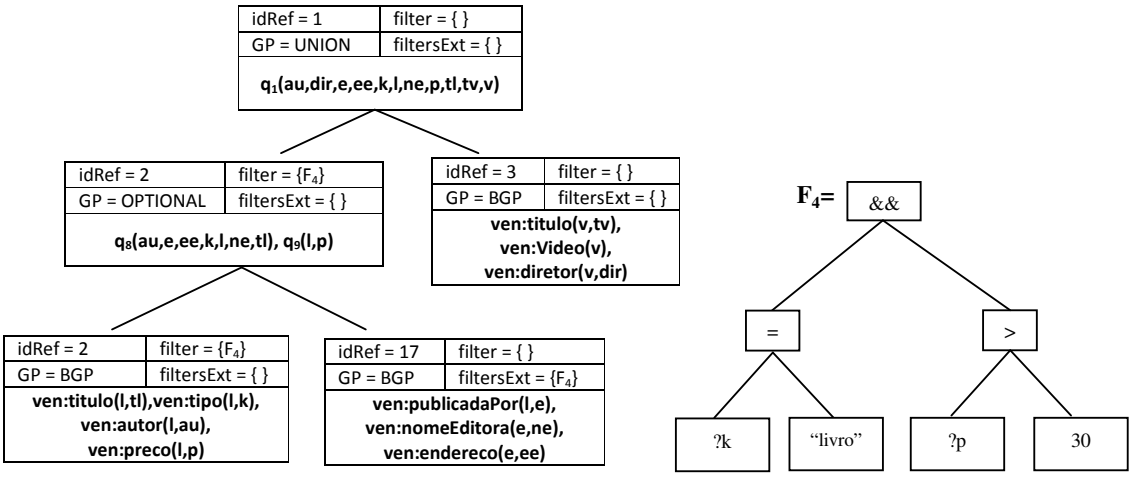


Figura 4.8. Árvore SPARQLD de representação da consulta do Exemplo 4.1

Exemplo de Construção de uma árvore SPARQLD

Como uma breve explanação da execução do *constroiSPARQLD*, considere as Figuras 4.6 e 4.8. Para este exemplo, *T.predicados* = $q_1(au,dir,e,ee,k,l,ne,p,tl,tv,v)$, sendo este o predicado que inicia a execução do procedimento *constroiDerivacoes*. A Tabela 4.1 exibe a execução de cada uma das chamadas recursivas efetuadas pelo algoritmo. Nesta tabela, *nA.predicados* consiste nos predicados existentes no *noAtual*, no início da execução; *p* é o predicado atual analisado; *u* é o número de unificações de *p* com a cabeças das regras; *nA.GP* é a indicação sobre o padrão de grafo que o nó representa.

Em #1, que corresponde a uma operação com o UNION, dois novos nós são criados, os quais são analisados nas chamadas #2 e #11, respectivamente. A chamada #2 efetua uma série de outras subchamadas, dentre as quais, uma que envolve o OPT (#5). Nela, também criados dois novos nós, que examinados nas chamadas #6 e #9,

respectivamente. O algoritmo encerra quando encontra indicações de BGPs (#8, #10,#12).

Exec	nA.predicados	p	u	nA.idRef	nA.GP	Chamadas de saída
#1	q_1(au,dir,e,ee,k,l,ne,p,tl,tv,v)	q_1(au,dir,e,ee,k,l,ne,p,tl,tv,v)	2 (regras 1 e 2)	1	UNION	#2 e #11
#2	q_2(au,e,ee,k,l,ne,p,tl)	q_2(au,e,ee,k,l,ne,p,tl)	1 (regra 3)	2		#3
#3	q_4(au,e,ee,k,l,ne,p,tl)	q_4(au,e,ee,k,l,ne,p,tl)	1 (regra 4)	2		#4
#4	F_4(((?k = "livro") && (?p > 30))), q_8(au,e,ee,k,l,ne,tl), q_9(l,p)	F_4(((?k = "livro") && (?p > 30)))		2		#5
#5	q_8(au,e,ee,k,l,ne,tl), q_9(l,p)	q_8(au,e,ee,k,l,ne,tl)	2 (regras 5 e 6)	2	OPT	#6 e #9
#6	q_{16}(au,k,l,tl), q_9(l,p)	q_{16}(au,k,l,tl)	1 (regra 7)	2		#7
#7	q_9(l,p), ven:titulo(l,tl), ven:tipo(l,k),ven:autor(l,au)	q_9(l,p)	1 (regra 9)	2		#8
#8	ven:titulo(l,tl),ven:tipo(l,k), ven:autor(l,au), ven:preco(l,p)	ven:titulo(l,tl)	0	2	BGP	
#9	q_{17}(e,ee,l,ne)	q_{17}(e,ee,l,ne)	1 (regra 8)	17		#10
#10	ven:publicadaPor(l,e), ven:nomeEditora(e,ne), ven:endereco(e,ee)	ven:publicadaPor(l,e)	0	17	BGP	
#11	q_3(dir,tv,v)	q_3(dir,tv,v)	1 (regra 10)	3		#12
#12	ven:titulo(v,tv),ven:Video(v), ven:diretor(v,dir)	ven:titulo(v,tv)	0	3	BGP	

Tabela 4.1. Exemplo de Execução do Algoritmo *ConstroiSPARQLD*

Após a realização dos procedimentos descritos nesta seção, a árvore SPARQLD de representação da consulta Q_{SPARQL} encontra-se construída. O processo de reescrita prossegue passando esta árvore como entrada para a próxima etapa, *Reescrita da Consulta e Reestruturação dos Resultados*.

4.5 Reescrita da Consulta e Reestruturação dos Resultados

Esta seção descreve o núcleo do processo de reescrita proposto neste capítulo, pois é através das tarefas desempenhadas pelo algoritmo aqui exibido que a consulta é, de fato, reformulada em termos da ontologia fonte. Para esta descrição, inicialmente, apresentamos algumas considerações sobre a representação das regras de mapeamento explanadas no capítulo anterior. Em seguida, explicamos o algoritmo *ReescritaSPARQLD*, que é responsável pela atividade de reescrita e reestruturação, bem como ilustramos um exemplo de execução deste algoritmo. Por fim, discutimos alguns aspectos relevantes a respeito desta etapa.

4.5.1 Representação das Regras de Mapeamento

Até então, as regras de mapeamento descritas foram apresentadas somente de maneira declarativa. No entanto, para que pudessem ser manipuladas pelo algoritmo proposto,

tais regras foram retratadas de acordo com o modelo ilustrado na Figura 4.9. Este modelo reflete a lei de formação das regras descrita no capítulo anterior.

Observe que esta lei de formação permite que cada conceito da ontologia alvo origine mais de uma regra de mapeamento. Com isso, agrupamos algumas regras com cabeças iguais em uma única regra com o corpo disjuntivo, visando tanto compactar o conjunto de regras quanto facilitar as manipulações envolvendo os contextos das propriedades, durante a reescrita. Este agrupamento das regras ocorre da seguinte forma:

(i) Para as classes, regras com cabeças iguais são reunidas em uma única regra.

(ii) Para as propriedades, regras com cabeças iguais e cujo corpo é diferenciado apenas pelo contexto também são agrupadas em uma única regra. Note que este caso corresponde à situação em que os contextos são subclasses do domínio (ou os *domínios restritos*) das propriedades presentes no corpo de todas as regras que serão reunidas.

Por exemplo, sejam as Figuras 3.6 e 3.7 (página 78 do Capítulo 3). Na Figura 3.6, há três regras (#12, #18 e #20) com a propriedade *v:titulo(x,y)* na cabeça. Além disso, no corpo destas três regras, os contextos são as classes *a:Livro*, *a:DVD* e *a:Musica*, respectivamente, as quais são subclasses das classe *a:Produto*, que é domínio de *a:descricao*. Já na Figura 3.7, uma situação semelhante ocorre para as regras #6 e #7. No entanto, neste caso, os contextos não são as subclasses do domínio da propriedade, mas sim o *domínio restrito*, que na prática, também representa um subconjunto do domínio, ou seja, uma subclasse implícita, semelhante à ideia de subclasse anônima em OWL. Observe que este tipo de contexto exemplificado reflete a expansão de uma hierarquia (ou de subconjuntos) de uma classe, a qual é domínio da propriedade mapeada.

Regras de Mapeamento		
Regra de Mapeamento	::=	Cabeça ← Corpo
Cabeça	::=	Classe | Propriedade
Corpo	::=	Conjunção_1 ; Conjunção_2 ; ... ; Conjunção_n
Conjunção	::=	Classe^+, Propriedade*, Contexto^+
Contexto	::=	$\text{Contexto na Ontologia Fonte}^+$, $\text{Restricão na Ontologia alvo}^+$
Contexto na Ontologia Fonte	::=	Classe^+, $\text{Restrição na Ontologia Fonte}^+$
Restrição na ontologia Fonte	::=	Propriedade, Átomo de operação (*built-in*)
Restrição na ontologia Alvo	::=	Átomo de operação (*built-in*)

Figura 4.9. Representação das Regras de Mapeamento

A título de esclarecimento, os símbolos presentes na Figura 4.9 possuem os seguintes significados: "|" indica que apenas um dos elementos aparece na cabeça da regra; "+" denota que o elemento do tipo descrito (Classe, por exemplo) aparece zero ou uma vez, enquanto "*" admite a ocorrência de tal elemento zero ou mais vezes; um átomo de operação (*built-in*) é um predicado com uma variável, um dos operadores já determinados na seção anterior e uma constante; uma conjunção é representada por uma vírgula; uma disjunção é uma lista de conjunções separadas por um ponto e vírgula.

Na prática, o modelo exibido na Figura 4.9 é implementado tendo como base as linguagens RULE ML[9] e SWRL[10], conforme será discutido no próximo capítulo (referente aos aspectos de implementação). A Figura 4.10 exibe um exemplo com as regras de mapeamento que serão empregadas durante a reescrita da consulta utilizada para ilustrações neste capítulo.

#3:	v:Video(p)	←	a:DVD(p)
#7:	v:tipo(p,y)	←	a:Livro(p), y= "livro"

#8:	v:titulo(p,y)	←	a:descricao(p,y), a:Livro(p);
			a:descricao(p,y), a:DVD(p);
			a:descricao(p,y), a:Musica(p)

#9:	v:preco(p,y)	←	a:preco(p,y), a:Livro(p);
			a:preco(p,y), a:DVD(p);
			a:preco(p,y), a:Musica(p)

#10:	v:autor(p,y)	←	a: autor(p,y), a:Livro(p)
#13:	v:nomeEditora(fEditora(y),y)	←	a:editora(p,y), a:Livro(p)
#14:	v:endereco(fEditora(y),z)	←	a:enderecoEditora(p,z), a:editora(p,y), a:Livro (p)
#15:	v:publicadaPor(p,fEditora(y))	←	a:editora(p,y), a:Livro(p)
#20	v:diretor(p,y)	←	a:temDiretor(p,z), a:nomeDiretor(z,y), a:DVD(p)

Legenda

——— Contexto

– – – – Restrição na ontologia fonte

..... Restrição na ontologia alvo

Figura 4.10. Exemplo de regras de mapeamento empregadas na reescrita.

[9] http://ruleml.org/

[10] http://www.w3.org/Submission/SWRL/

4.5.2 O Algoritmo *ReescritaSPARQLD*

O algoritmo *ReescritaSPARQLD,* o qual é exibido na Figura 4.11 e auxiliado pelos algoritmos *Substituição* (Figura 4.12) e *CompatibilizaContexto* (Figura 4.13), recebe como entrada a árvore SPARQLD de representação da consulta (construída da etapa anterior) e um conjunto de regras de mapeamento (M_{ST}) entre as ontologias fonte e alvo. O objetivo deste algoritmo consiste em complementar a árvore SPARQLD com as derivações obtidas por meio destas regras, de forma que tal árvore retrate a consulta reescrita. Além disso, este algoritmo produz dois artefatos adicionais: (i) um conjunto de atualizações sobre R de Q_N (onde R é o formato de retorno da consulta), as quais possibilitam que o retorno da consulta possa ser apresentado de acordo com a ontologia alvo; (ii) um conjunto com as indicações de subárvores que devem ser desconsideradas da consulta reescrita.

Mais precisamente, para atingir o seu objetivo, o *ReescritaSPARQLD* funciona através dos passos enumerados abaixo. Na nomenclatura utilizada, o termo *resolver* diz respeito a exprimir os conceitos da consulta em termos da ontologia fonte, utilizando, para isto, as regras de mapeamento. Assim, um predicado *resolvido* é um predicado que expressa um conceito da ontologia fonte.

Funcionamento Geral do *ReescritaSPARQLD*

1. Inicializa os conjuntos de saída (linhas 1-2).

Estes conjuntos possuem escopo global, ou seja, para toda a consulta (toda a árvore). Assim, as modificações que sofrem ao longo do processo são mantidas até a finalização do algoritmo.

2. Para cada nó folha da árvore, é realizada uma chamada ao procedimento *Resolva*, com o intuito de gerar as derivações vinculadas a este nó (linhas 3-5).

Cada folha de uma árvore SPARQLD contém uma conjunção de predicados, o que representa uma consulta conjuntiva *CQ*. Os operadores que associam estas *CQs* encontram-se distribuídos ao longo da árvore.

3. Uma vez obtidas as derivações relativas a cada uma das folhas, a árvore SPARQLD final e os conjuntos de *termos atualizados* e *ramos eliminados* são retornados (linha 6).

ENTRADA: SPARQLD inicial, Ontologia alvo, Mapeamentos M_{ST}
SAÍDA: SPARQLD da consulta reescrita.
Conjunto com os termos que precisam ser atualizados no conjunto *R* de Q_N.
Conjunto com os ramos que não dever ser considerados na consulta reescrita

ReescritaSPARQLD (SPARQLD Q, M_{ST})

```
1.  termosAtualizados := ∅;
2.  ramosEliminados :=  ∅;
3.  Para cada folha f de Q Faça
4.     Resolva(f, M_ST);
5.  Fim Para
6.  Retorne Q, termosAtualizados,ramosEliminados;
```

Fim *reescritaSPARQLD*

Resolva(noCQ, M) // cada nó folha contém uma consulta conjuntiva

```
7.  Se noCQ.predicados.inicio não resolvido ou noCQ.predicados <>  vazio Então
8.  predicadoAtual := noCQ.predicados.inicio;
9.  UCQPredicadosAux :=∅;
10.   No caso
11.     UCQ_Rule := ∅;
12.    /********************** Classe*****************************/
13.    Caso 1: Se predicadoAtual é da forma C(t) Então
14.        Se as unificações com C(t) ainda não foram realizadas Então
15.            Para cada unificao(C(t), C(t´)) com  alguma regra do tipo C'(t') ← Φ1; Φ2; ... ; Φn em M Faça
16.                UCQ_Rule := conjunto de conjunções retorno de substituicao(θ, {Φ1, Φ2, ... ,Φn}, termosAtualizados);
17.                geraConsultaClasse(UCQ_Rule);
18.            Fim Para
19.        Senão
20.            Para cada unificao(C(t), C(t´)) já realizada com  alguma regra do tipo C'(t') ← Φ1; Φ2; ... ; Φn em M Faça
21.                UCQ_Rule := substituição já efetuada utilizando C(t) e o corpo da regra em M;
22.                geraConsultaClasse(UCQ_Rule);
23.            Fim Para
24.        Fim Se
25.    /********************** Propriedade*************************/
26.    Caso 2: Se predicadoAtual é da forma p(t1,t2) Então
27.        Se as unificações com p(t1,t2) ainda não foram realizadas Então
28.            Para cada unificao(p(t1,t2), p(t1',t2')) com alguma regra do tipo p(t1',t2')← Φ1; Φ2; ... ; Φn em M Faça
29.                UCQ_Rule := conjunto de conjunções retorno de substituicao(θ, {Φ1, Φ2, ... ,Φn}, termosAtualizados););
30.                geraConsultaPropriedade(UCQ_Rule);
31.            Fim Para
32.        Senão
33.            Para cada unificao(p(t1,t2), p(t1',t2')) já realizada com alguma regra do tipo p(t1',t2')← Φ1; Φ2; ... ; Φn em M Faça
34.                UCQ_Rule := substituição já efetuada utilizando p(t1,t2) e o corpo da regra em M;
35.                geraConsultaPropriedade(UCQ_Rule);
36.            Fim Para
37.        Fim Se
38.    Fim Caso
39.    /**********************Preenche a Árvore*************************/
40.    Se UCQPredicadosAux = ∅  Então
41.        Adiciona noCQ.idRef em ramosEliminados //indica falha do ramo;
42.    Fim Se
43.    Se UCQPredicadosAux.tamanho = 1 Então
44.        noCQ.GP = BGP;
45.        noCQ.predicados := UCQPredicadosAux;
46.        Resolva(noCQ);
47.    FimSe
48.    Se UCQPredicadosAux.tamanho > 1 Então
49.        noCQ.GP = UNION;
50.        Para cada predicadosAux i em UCQPredicadosAux Faça
51.            novoNo = CriarNoSPARQLD;
52.            novoNo.idRef := inteiro ainda não utilizado;
53.            Adicionar novoNo como filho de noCQ;
54.            Resolva(novoNo);
55.        Fim Para
```

```
56.     Fim Se
57.   Fim Se
Fim Resolva

geraConsultaClasse(UCQ UCQ_Rule)
58.   Para cada conjunção predicadosCQ_Rule em UCQ_Rule Faça
59.       predicadosAux:= noCQ.predicados – {C(t)};
60.       Adicione predicadosCQ_Rule em predicadosAux;
61.       Adicione predicadosAux em UCQPredicadosAux;
62.   Fim Para
Fim geraConsultaClasse

geraConsultaPropriedade(UCQ UCQ_Rule)
63.   Para cada conjunção predicadosCQ_Rule em UCQ_Rule Faça
64.       predicadosAux:= noCQ.predicados – {p(t1,t2)};
65.       Se não há predicados resolvidos em predicadosAux Então  //para isso, verifica se o último termo da lista está resolvido
66.           Adicione predicadosCQ_Rule em predicadosAux;
67.           Adicione predicadosAux em UCQPredicadosAux;
68.       Senão
69.           /******** Evita a geração ou elimina ramos que representem a reescrita de consultas
70.                       redundantes, que não correspondam à original ou que retornariam vazio ***********/
71.           Se compatibilizaContexto(predicadosAux, predicadosCQ_Rule) é verdadeiro Então
72.               Adicione predicadosAux em UCQPredicadosAux;
73.               Sai do laço Para;  //evita várias ramificações que seriam redundantes ou vazias (contexto expandido)
74.           Fim Se
75.       Fim Se
76.   Fim Para
geraConsultaPropriedade
```

Figura 4.11. Algoritmo *ReescritaSPARQLD*

Funcionamento do procedimento *Resolva*

Este procedimento, que recebe como entrada um nó da árvore (*noCQ*) e um conjunto (*M*) de mapeamentos entre as ontologias, é executado até que todos os predicados do *noCQ* estejam resolvidos ou o conjunto destes predicados esteja vazio (teste da linha 7).

1. Verifica se o predicado atual (primeiro da lista de predicados) é uma classe ou uma propriedade (linhas 13 e 25)**.**

2. Averigua se este predicado ainda não foi buscado em *M* (linha 14).

Salientamos que este teste tem o intuito apenas de evitar que o mesmo predicado faça, diversas vezes, a mesma busca no conjunto *M*. Assim, após a primeira busca, a regra é armazenada para futuras utilizações.

3. Se o predicado for uma classe, obtém, em *M*, todas as regras do tipo $C(t') \leftarrow \Phi_1; \Phi_2; \ldots ; \Phi_n$ com as quais *C(t)* unifica com *C(t')* (linha15). Chamaremos estas regras, de agora em diante, de regras selecionadas.

Para cada unificação, é gerado um *mgu* θ. Este *mgu* representa o conjunto de substituições que devem ser realizadas em cada Φ_K.

4. Invoca a função *Substituição* (algoritmo da Figura 4.12), visando trocar cada uma das variáveis presentes no corpo da regra selecionada pelas variáveis e/ou constantes do *predicadoAtual* (linha 16).

Na chamada para esta função, são passados: o *mgu* Θ entre C(t) e C(t'); o corpo da regra selecionada, o qual contém uma disjunção de conjunções (cada conjunção é representada por Φ); o conjunto de *termos atualizados*, pois este conjunto pode sofrer alterações durante a execução de *substituição*.

5. Chama o procedimento *geraConsultaClasse* (linha 22).

6. Se o predicado atual for uma propriedade, as ações desempenhadas são semelhantes às realizadas paras as classes (linhas 27 a 37), exceto pelo procedimento *geraConsultaPropriedade*, que faz uso das informações sobre os contextos oriundos das regras.

7. Preenche a árvore (linhas 40 a 56). *UCQPredicadosAux* é um conjunto (inicializado na linha 9) preenchido durante os procedimentos *geraConsultaClasse* e *geraConsultaPropriedade*. Este conjunto contém uma disjunção de conjunções (representa uma união de consultas conjuntivas) de forma que, de acordo com o conteúdo dele, a árvore SPARQLD é preenchida da seguinte maneira:

(i) Se ele for vazio, significa que nenhuma consulta foi gerada e, portanto, o ramo é falho. Isto pode acontecer por um dos seguintes motivos: (i) não há correspondência para o *predicadoAtual* no conjunto M_{ST}, o que exprime que este conceito não está disponível na ontologia fonte; (ii) há regras correspondentes em M_{ST}, mas a consulta obtida foi descartada durante o procedimento c*ompatibilizaContexto*. Em ambos os casos, há uma falha no ramo. (linhas 40-42)

(ii) Se ele possui apenas um elemento, não há disjunções, não sendo necessária a criação de um novo nó. Os predicados do nó atual são apenas substituídos pela conjunção existente em *UCQPredicadosAux*. (linhas 43 a 47).

(iii) Se ele possui mais de um elemento, há disjunções. Assim, o nó é anotado como sendo uma união e novos nós filhos são criados para cada uma das consultas conjuntivas. (linhas 48 a 56)

Funcionamento dos procedimentos *geraConsultaClasse* e *geraConsultaPropriedade*

Em suma, estes procedimentos retiram o *predicadoAtual* da consulta, uma vez que ele já foi resolvido, e inserem os predicados obtidos a partir das regras. Cada conjunção resultante desta inserção origina uma consulta conjuntiva que é armazenada em *UCQPredicadosAux*. No caso das propriedades, o procedimento *geraConsultaPropriedade* utiliza a função c*ompatibilizaContexto* (algoritmo da Figura 4.13) para verificar se tais consultas conjuntivas devem ser realmente geradas (linhas 65 a 76). Remetemos o leitor a esta função e aos exemplos descritos na próxima seção (subseção 4.5.3) para entender melhor como estas atividades acontecem.

ENTRADA: Um conjunto de substituições Θ, que consiste no *mgu* de C(t) e C(t´) ou de $p(t_1,t_2)$ e $p(t_1',t_2')$. Um conjunto UCQ com as conjunções presentes no corpo da regra. O conjunto T com as modificações de termos que necessitam ser guardadas. **SAÍDA:** Um conjunto UCQ', resultado das substituições e alterações realizadas em UCQ Conjunto T atualizado.	
Substituicao (mgu Θ, Conjunto UCQ, Conjunto T)	
1.	/** *As variáveis presentes em* UCQ *são substituídos pelo termo t de C(t)* ou por t_1,t_2 de $p(t_1,t_2)$ *, exceto para as funções***/
2.	Seja UCQ = { Φ_1, $\Phi_{2,\ldots}$ Φ_n }, onde Φ é uma conjunção de termos.
3.	UCQ' := UCQΘ = { $\Phi_1\Theta$, $\Phi_2\Theta_{,\ldots,}\Phi_n\Theta$}; //instância de UCQ por Θ
4.	/*** Atualiza T utilizando as substituições contidas em Θ ****/
5.	Seja T = {u_1/m_1, ..., u_n/m_n}, onde u_i deve ser uma variável presente na consulta original e m_i uma função que deverá substituí-la na reestruturação.
6.	**Para cada** *s/t* em Θ **Faça**
7.	**Se** *t* é uma função **Então**
8.	Insira s/*t* em T;
9.	**Senão**
10.	Atualize *T* de acordo com *s/t*, se necessário.
11.	**Fim Se**
12.	**Fim Para**
13.	Renomeie, em UCQ', as variáveis de UCQ que não sofreram mudanças por variáveis ainda não utilizadas.
14.	**Retorne** UCQ', T**;**
Fim *substituicao*.	

Figura 4.12. Algoritmo *Substituição,* utilizado pelo *ReescritaSPARQLD*

Funcionamento Geral do *Substituição*

A Figura 4.12 apresenta o algoritmo *Substituição*, o qual tem como função trocar as variáveis contidas no corpo da regra selecionada pelos termos (constante ou variáveis) presentes no *predicadaAtual* da consulta. Esta substituição é realizada utilizando o *mgu* Θ (linha 3). As linhas 4 a 12 correspondem ao caso onde, no *mgu*, há um elemento do tipo *v/f(x)*, ou seja, uma indicação de que uma função deve substituir uma variável. Isto ocorre quando há funções na cabeça da regra de mapeamento. Conforme explanado no Capítulo 3, estas funções são utilizadas para expressar a reestruturação de informação (Casos 5 e 6 das regras de mapeamento) na situação onde uma classe da ontologia alvo

está representada como uma propriedade na ontologia fonte. Por este motivo, armazenamos esta substituição em *T* (linhas 6 a 11). A ideia é, posteriormente, utilizarmos esta informação para reestruturar o retorno da consulta de acordo com a ontologia alvo. Finalmente, a linha 13 troca as variáveis (do corpo da regra selecionada) que não sofreram mudanças por outras ainda não utilizadas, visando evitar associações incorretas na consulta.

ENTRADA: Lista com os predicados atuais da consulta, Lista com predicados que devem ser inseridos na consulta.
SAÍDA: *pQuery* atualizada, se os predicados obtidos das regras estiverem aptos a compor a consulta reescrita. *Falha*, caso contrário.

CompatibilizaContexto (Lista de predicados da consulta pQuery, Lista de predicados do corpo da regra pRule)

```
1.  Seja contextRule o contexto de pRule
2.  inseridoAux := falso;
3.  /****************** Restrição na Ontologia Alvo*********************/
4.  Se há Restrição na Ontologia Alvo em contextRule Então
5.  Seja builtRestT o átomo de operação contido nesta restrição.
6.     Para cada ocorrência de var(builtRestT) nos filtros do noCQ Faça
7.        Marcar operação para ser removida do filtro;
8.        Se verficaOperação(operacaoFiltro, valorFiltro, operacao(builtRestS), valor(builtRestS)) é falso Então
9.           Retorne falso;
10.       Fim Se
11.    Fim Para
12. Fim Se
13. /**********************Restrição na Ontologia Fonte*********************/
14. Se há Restrição na Ontologia Fonte em contextRule Então
15.   //Propriedade
16.    Seja propRest a propriedade contida na restrição.
17.    Se propRest unifica com alguma propriedade de pQuery Então
18.       reduzConsulta(propRest);
19.    Senão
20.       Insere propRest em pQuery;
21.    Fim Se
22.   //Operação
23.    Seja builtRestS o átomo de operação contido nesta restrição.
24.    Para cada átomo de built-in builtQuery em pQuery que possua var(builtQuery) = var(builtRestS) Faça
25.       Se verficaOperaco(operacao(builtQuery), valor(builtQuery), operacao(builtRestS), valor(builtRestS)) é Verdadeiro Então
26.          inseridoAux :=  verdadeiro;
27.       Senão
28.          Retorne Falha;
29.       Fim Se
30.    Fim Para
31.    Se inseridoAux = falso Então
32.       Insere builtRestS em pQuery;
33.    Fim Se
34. Fim Se
35. /**********************Classe do Contexto*****************************/
36. Seja contextClass a classe pertencente ao Contexto na Ontologia Fonte
37.  inseridoAux :=  falso;
38. Para cada predicado resolvido pr em pQuery Faça
39.    Se contextClass unifica com pr Então
40.       reduzConsulta(pr,contextClass);
41.       inseridoAux := verdadeiro;
42.    Senão
43.       Se conceitos (símbolo de predicado) de contextClass e pr são diferentes E os termos iguais Então
             Retorne Falha;
44.       Fim se
45.    Fim Se
46. Fim Para
47. Se inseridoAux = falso Então
48.    Insere contextClass em pQuery;
```

```
49.  Fim Se
50.  /******************Demais predicados (não contexto) da Conjunção**************/
51.  Para cada predicado pnContext não pertencente ao Contexto em pRule Faça
52.  inseridoAux = falso;
53.     Para cada predicado resolvido pr em pQuery Faça
54.        Se pnContext unifica com pr Então
55.           reduzConsulta(pr,pnContext);
56.           inseridoAux := verdadeiro;
57.        Senão
58.           Se pr é contexto E conceitos (símbolo de predicado) de pr e pnContext são diferentes E os termos iguais Então
59.              Retorne Falha;
60.           Fim se
61.        Fim Se
62.     Fim Para
63.     Se inseridoAux = falso Então
64.        Insere pnContext em pQuery;
65.     Fim Se
66.  Fim Para
67.  Retorne pQuery;
```

Fim *compatibilizaContexto*.

Nota: O procedimento *reduzConsulta* mantém apenas um dos dois predicados na consulta, de acordo com a unificação, priorizando as variáveis da consulta original do usuário. Além disso, se necessário, realiza as atualizações dos termos restantes em *pRule*.

Figura 4.13. Algoritmo *CompatibilizaContexto*, utilizado pelo *ReescritaSPARQLD*

Funcionamento Geral do *CompatibilizaContexto*

O algoritmo *CompatibilizaContexto*, exibido na Figura 4.13, recebe como entrada: (i) uma lista com os predicados existentes na consulta (*pQuery*), isto é, aqueles que já foram resolvidos; (ii) uma lista com os predicados que devem ser inseridos na consulta (*pRule*), ou seja, aqueles obtidos através das regras de mapeamento, em uma dada chamada. O objetivo deste algoritmo consiste em verificar se a consulta resultante da conjunção entre *pRule* em *pQuery* será uma consulta que, a priori, sabemos que retornará vazio e, portanto, pode ser descartada. Esta verificação é realizada utilizando as informações de contexto oriundas das regras. Para isto, o algoritmo averigua se não há "incompatibilidades" entre:

1. Cada um dos predicados que compõem os elementos do contexto da regra e os predicados já existentes na consulta (linhas 3 a 49).
2. Os predicados que não são contexto (na regra) e os predicados da consulta que tenham o rótulo de contexto (50 a 66).

Note que as averiguações das linhas 3 a 12 e 13 a 34 são executada para as regras relativas, respectivamente, aos Casos 2 e 3 no Capítulo 3. O restante do algoritmo é executado para todos os tipos de regras. Além disso, o algoritmo *CompatibilizaContexto* evita a inserção de predicados repetidos, através do procedimento *reduzConsulta*. Como saída, este algoritmo retorna *falso*, se houver alguma incompatibilidade. Caso contrário, retorna a consulta conjuntiva pronta para ser

inserida no nó. Observe que o descarte possibilita a descontinuidade na execução de um nó que não contribuirá para a consulta.

Os exemplos da próxima seção auxiliam no entendimento dos passos descritos.

4.5.3 Exemplo de Execução do *ReescritaSPARQLD*

Exemplo 1

A Figura 4.14 exibe a execução do algoritmo para a consulta com a qual estamos trabalhando durante este capítulo (Exemplo 4.1), ou seja, *Q = Recupere o título e o autor de todos os livros que custam mais de 30 reais. Caso as informações sobre a editora vinculada ao livro estejam disponíveis, recupere tal editora, juntamente com o nome e o endereço da mesma. Além disso, retorne o título de todos os Vídeos, com seus respectivos diretores. Ordene os resultados pelo título*. Esta consulta utiliza grande parte das operações desempenhadas pelos algoritmos apresentados anteriormente.

Visando facilitar o entendimento do processo de reescrita desta consulta, na Figura 4.14, temos que (observe a legenda): em cada nó, o *predicado atual* está sublinhado; os nós verdes correspondem à árvore SPARQLD de entrada; os nós com setas retratam as saídas do algoritmo. As Tabelas 4.2, 4.3 e 4.4 ilustram a execução das chamadas recursivas para cada uma das folhas da árvore original de entrada. Note que tais folhas podem se ramificar ao longo da execução do algoritmo.

- **Folha 1**

Chamada #1: Nesta chamada, *predicadoAtual = ven:titulo(l,t)*. Com isso, o algoritmo entra no Caso 2 (linha 26). Como este predicado ainda não foi buscado no conjunto M_{ST} das regras de mapeamento, tal busca é realizada. A cabeça da regra 8 (*v:preco(p,y)*) unifica com o *predicadoAtual* e o *mgu* retornado é θ = {p/l, y/t}, ou seja, *l* e *t* devem ser trocados por *p* e *y*, respectivamente. Dessa forma, na linha 29, a função ***Substituição*** é invocada com os seguintes valores de entrada: θ = {p/l, y/t}, UCQ = {[a:descricao(p,y), a:Livro(p)], [a:descricao(p,y), a:DVD(p)],[a:descricao(p,y), a:Musica(p)]} e T = .

Dentro desta função, a substituição é efetuada e UCQ' = {[a:descricao(l,tl), a:Livro(l)], [a:descricao(l,tl), a:DVD(l)],[a:descricao(l,tl), a:Musica(l)]}. Como não há funções no *mgu*, ou seja, a regra 8 não indica reestruturação, então o algoritmo não executa as rotinas das linhas 7 a 13. Além disso, todas as variáveis de UCQ foram substituídas por variáveis vindas da consulta do usuário, portanto, não há necessidade

de trocá-las (linha 14).

De volta à rotina principal, *UCQ_Rule* recebe o retorno de *Substituicao*, ou seja, UCQ'. Assim, na linha 30, o procedimento ***geraConsultaPropriedade*** é invocado e recebe *UCQ_Rule* como entrada. Neste procedimento, *predicadosAux* = {ven:tipo(l,k), ven:autor(l,au), ven:preco(l,p)}, isto é, os predicados do nó, excluindo o *predicadoAtual*. Na linha 65, o algoritmo verifica se há predicados resolvidos em *predicadosAux*. Este teste tem o seguinte objetivo: caso houvesse predicados resolvidos, o algoritmo iria checar, para cada conjunção de *UCQ_Rule*, se tal conjunção poderia ser inserida, sem que fosse gerada uma consulta redundante ou de retorno vazio. Como não há predicado resolvidos, *UCQPredicadosAux* = {[ven:tipo(l,k), ven:autor(l,au), ven:preco(l,p), a:descricao(l,tl), a:Livro(l)], [(ven:tipo(l,k), ven:autor(l,au), ven:preco(l,p), a:descricao(l,tl), a:DVD(l))], [(ven:tipo(l,k), ven:autor(l,au), ven:preco(l,p), a:descricao(l,tl), a:Musica(l))]}.

Finalmente, a árvore pode ser preenchida. Como *UCQPredicadosAux.tamanho* = 3 (número de consultas conjuntivas), o algoritmo entra na linha 47 e executa as seguintes ações: (i) anota o nó como sendo uma união (linha 49); (ii) cria um nó para cada um dos elementos do conjunto *UCQPredicadosAux*; (iii) chama o algoritmo para estes nós criados. Tais nós criados podem ser vistos na Figura 4.14 (nós brancos com *idRefs* iguais a 18, 19 e 20, respectivamente).

	Folha 1 (idRef = 2)						
	noCQ. idRef	**Predicado Atual**	**Regras de *M* utilizadas**	**UCQ_Rule**	**Termos Atualizados**	**Ramos Eliminados**	**Chamadas de Saída**
#1	2	ven:titulo(l,t)	8	a:descricao(l,tl), a:Livro(l); a:descricao(l,tl), a:DVD(l); a:descricao(l,tl), a:Musica(l)	Ø	Ø	#2, #6, #7
#2	18	ven:tipo(l,k)	7	a:Livro(l), k = "livro"	Ø	Ø	#3
#3	18	ven:autor(l,au)	10	a:autor(l,au), a:Livro(l)	Ø	Ø	#4
#4	18	ven:preco(l,p)	9	a:preco(l,p), a:Livro(l); a:preco(l,p), a:DVD(l); a:preco(l,p), a:Musica(l)	Ø	Ø	#5
#5	18	a:descrição(l,tl)	-	-	Ø	Ø	-
#6	19	ven:tipo(l,k)	7(*)	a:Livro(l), k = "livro"	Ø	{19}	-
#7	20	ven:tipo(l,k)	7(*)	a:Livro(l), k = "livro"	Ø	{19, 20}	-

Tabela 4.2. Exemplo de execução do Algoritmo *ReescritaSPARQLD* (Folha 1)

Como muitos dos passos do algoritmo já foram descritos na Chamada #1, nas próximas exemplificações, focaremos em aspectos distintos e específicos de cada uma delas.

Chamada #2: Nesta chamada, *predicadoAtual* = *ven:titulo(l,t)*, de maneira que a regra 7 é selecionada em M_{ST}. Observe (na Figura 4.10) que tal regra contém uma restrição na ontologia alvo. Durante a execução do procedimento ***geraConsultaPropriedade***, *UCQ_Rule* = {[a:Livro(l), k = "livro"]}. Como *predicadosAux* = {[ven:autor(l,au), ven:preco(l,p), a:descricao(l,tl), a:Livro(l)]}, o teste da linha 65, constata que há predicados resolvidos no nó, os quais foram inseridos na Chamada #1. Com isso, a função ***CompatibilizaContexto*** é invocada, visando verificar se a conjunção presente em *UCQ_Rule* é "compatível" com o restante da consulta existente.

Neste momento, a função ***CompatibilizaContexto*** recebe como entrada *pQuery* = {ven:autor(l,au), ven:preco(l,p), a:descricao(l,tl), a:Livro(l)}e *pRule* = {a:Livro(l), k = "livro"}. Esta função inicia averiguando a restrição, na ontologia alvo, *k = livro* (linhas 3 a 12). Recordamos o leitor de que uma restrição foi definida, anteriormente, como sendo uma tripla (*propriedade, operação, valor*). Portanto, para que a regra com tal restrição tenha sido selecionada, é necessário que a consulta do usuário contenha a propriedade que denota a restrição. Neste caso, esta propriedade é *tipo(l,k)*. Sendo assim, o algoritmo verifica se, na consulta, o valor atribuído à propriedade *tipo* é compatível com o valor presente na regra (linha 8). Para esta execução, a verificação é verdadeira. No entanto, observe que se, no filtro, houvesse o valor *k = Musica*, o retorno seria falso. Por outro lado, a linha 7 indica que operação deve ser removida do filtro, pois ela se aplica à uma propriedade que tem como único objetivo delimitar um subconjunto na ontologia alvo. Finalmente, os testes para os demais predicados de *pRule* (*a:Livro(l)*) também são compatíveis.

Observe que nesta chamada, não há a criação de um novo nó, mas apenas uma substituição dos predicados deste nó, uma vez que não há disjunções.

Chamadas #3 e #4: De modo geral, ocorrem de maneira análoga às anteriores.

Chamada #5: Critério de parada do algoritmo. Não passa no teste inicial (linha 7), sinalizando que a consulta do nó já se encontra toda reescrita.

Chamada #6: Corresponde a um dos nós (*idRef* = 19) gerados como saída da Chamada #1. A particularidade desta execução está no fato deste ser um nó falho. Isto é constatado em um dos testes da linha 58 do algoritmo ***CompatibilizaContexto***. Nele, temos que *DVD(l)*, predicado de contexto inserido na Chamada #1, não é compatível com *Livro(l)*, pertencente à *pRule*. A conjunção destes dois conceitos exprimiria que *l* é, simultaneamente, uma instância de *Livro* e *DVD*. Para entender o porquê desta consulta

ser redundante ou vazia, observe a árvore da Figura 4.14. Cada um dos nós 18, 19 e 20 foi gerado a partir de *titulo(l,t)*, através de uma disjunção no corpo da regra. Tal disjunção representa a expansão de uma hierarquia. Assim, se *l* for uma instância de *Livro* e *DVD*, ele já estará sendo recuperado na consulta do nó de *idRef* = 18 (redundância). Caso, contrário, *Livro(l)* e *DVD(l)* gerariam uma consulta vazia. Por fim, como este nó pode ser desconsiderado, na rotina principal do algoritmo (linha 41), ele é adicionado no conjunto de *Ramos Eliminados* e sua derivação não é mais executada.

Chamada #7: Semelhante à #6.

Note que, em todas as execuções, não há inserção de predicados repetidos e as variáveis originais da consulta são mantidas.

- **Folha 2:**

Chamada #1: Nesta chamada, *predicadoAtual* = *nomeEditora(e,ne)*, o qual unifica com *nomeEditora(fEditora(y),y)*. Com isso, o *mgu* obtido é Θ = {y/ne, e/ *fEditora(ne)*}. Observe que esta chamada ilustra o caso onde a cabeça da regra selecionada (regra 15) possui uma função. Esta situação acontece porque a classe *Editora* não existe na ontologia fonte. Sendo assim, a variável *e*, que corresponde ao identificador de um objeto do tipo *Editora*, não pode ser diretamente recuperada. Para que esta variável possa receber algum valor, é necessário construir tal instância, a partir dos valores disponíveis na fonte. Para este fim, a função *fEditora(ne)* deve ser utilizada. É por este motivo que, na Tabela 4.3, o conjunto *Termos Atualizados* armazena [e/*fEditora(ne)*](linhas 6 a 12 da função *Substituição*).

	Folha 2 (idRef = 17)						
	noCQ. idRef	**Predicado Atual**	**Regras de *M* utilizadas**	**UCQ_Rule**	**Termos Atualizados**	**Ramos Eliminados**	**Chamadas de Saída**
#1	17	ven:nomeEditora(e,ne)	15	a:editora(_p13,ne), a:Livro(_p13)	{e/*fEditora*(ne)}	{19, 20}	#2
#2	17	ven:endereco(e,ee)	13	a:enderecoEditora(_p14,ee),a:editora (_p14,_ne), a:Livro(_p14)	{e/*fEditora*(ne)}	{19, 20}	#3
#3	17	ven:publicadaPor(l,e)	14	a:editora(l,ne), a:Livro(l)	{e/*fEditora*(ne)}	{19, 20}	#4
#4	17	a:editora(e,ne)	-	-	{e/*fEditora*(ne)}	{19, 20}	-

Tabela 4.3. Exemplo de execução do Algoritmo *ReescritaSPARQLD* (Folha 2)

Chamadas #2, #3 e #4: Similares à Chamada #1. Destacamos apenas que, como o *predicadoAtual* destas chamadas refere-se à mesma função (*fEditora*) da anterior, é importante ter atenção com as correspondências entre as variáveis, visando evitar ligações equivocadas. Por fim, observe que as variáveis iniciadas "_"simbolizam as variáveis renomeadas (linha 13 da função *Substituição*).

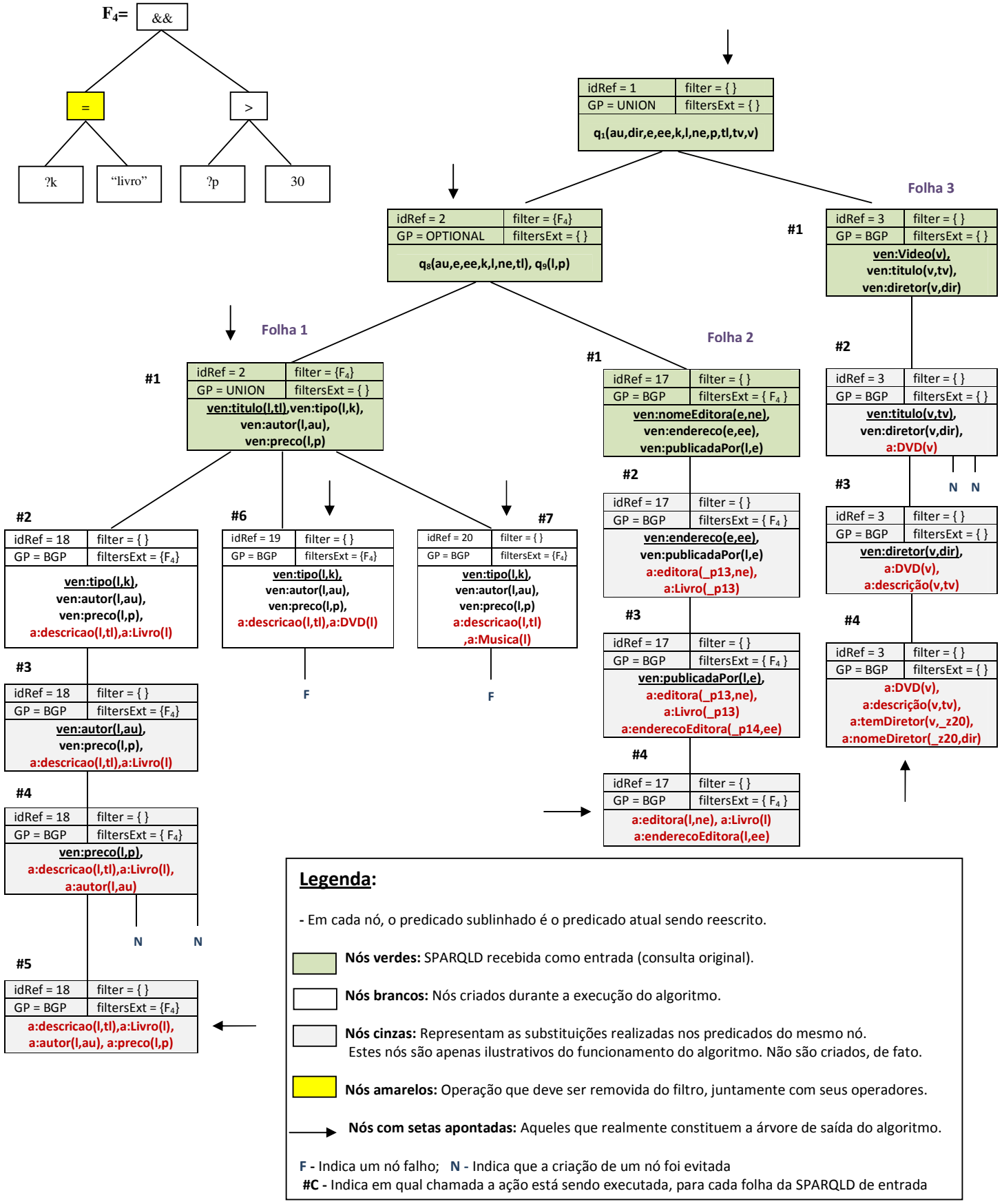


Figura 4.14. Exemplo de execução do algoritmo *ReescritaSPARQLD* para a consulta do Exemplo 4.1

Folha 3:

A execução é análoga às chamadas descritas até então. Nesta folha, destacamos apenas a Chamada #2. Nela, o teste da linha 71 a 75, do procedimento *geraConsultaPropriedade,* evita a criação de novos ramos para as conjunções [*a:descricao(v,tv), a:Livro(v)*] e [*a:descricao(v,tv), a:Musica(v)*].

	Folha 3 (idRef = 3)						
	noCQ. idRef	**Predicado Atual**	**Regras de *M* utilizadas**	**UCQ_Rule**	**Termos Atualizados**	**Ramos Eliminados**	**Chamadas de Saída**
#1	3	ven:Video(v)	3	a:DVD(v)	{e/*fEditora*(ne)}	{19, 20}	#2

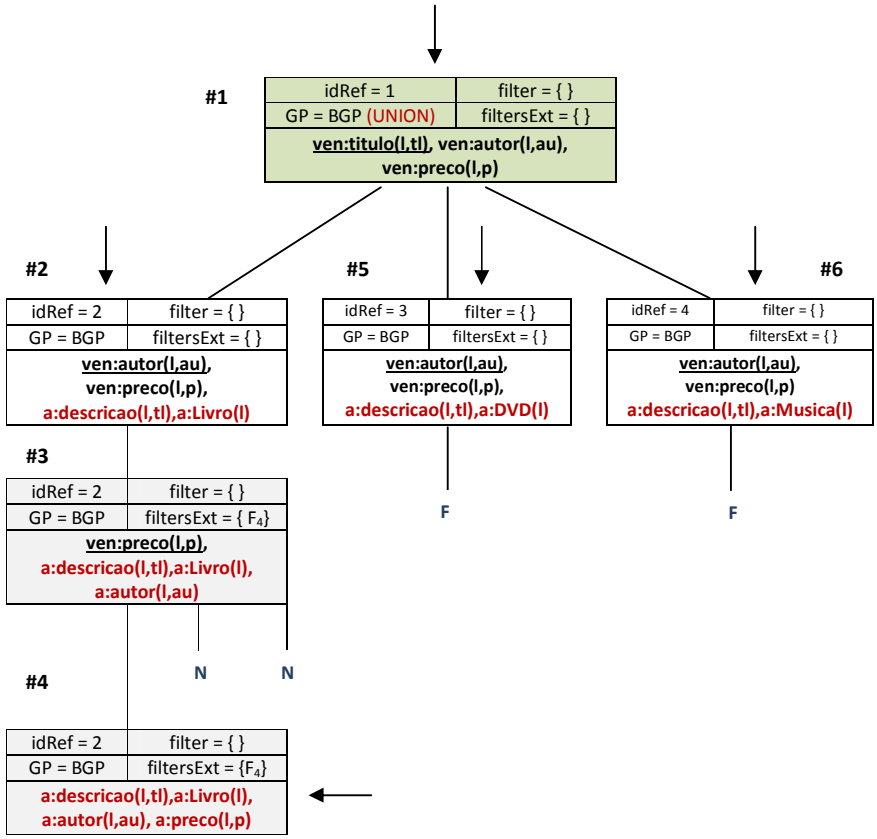
#2	3	ven:titulo(v,tv)	8	a:descricao(v,tv), a:Livro(v);
				a:descricao(v,tv), a:DVD(v);
				a:descricao(v,tv), a:Musica(v);
					{e/*fEditora*(ne)}	{19, 20}	#3

#3	3	ven:diretor(ven,dir)	20	a:temDiretor(v,_z20),a:nomeDire tor(_z20,dir), a:DVD(v)	{e/*fEditora*(ne)}	{19, 20}	#4
#4	3	a:DVD(v)	-	-	{e*fEditora*(ne)}	{19, 20}	-

Tabela 4.4. Exemplo de execução do Algoritmo *ReescritaSPARQLD* (Folha 3)

Exemplo 2

Neste exemplo, ilustramos a execução da seguinte consulta: *Q = Recupere o título, o autor e preço de todos os Livros.* O nó verde representa a árvore de entrada (note que *Q* é simplesmente uma conjunção) e os nós com as setas, as saídas.



4.15. Exemplo de execução do algoritmo *ReescritaSPARQLD* para a consulta *Q* ilustrada

Na **Chamada #1**, o predicado *ven:titulo(l,tl)*, gera três novos ramos, correspondentes aos contextos *Livro(l)*, *DVD(l)* e *Musica(l)*. As **Chamadas #5** e **#6** descartam a continuidade dos nós de *idRef* = 3 e 4. Isto porque *ven:autor* é uma informação presente em *Livro*, mas não em *DVD* e *Musica*.

	Folha 1 (idRef = 1)						
	noCQ. idRef	Predicado Atual	Regras de *M* utilizadas	UCQ_Rule	Termos Atualizados	Ramos Eliminados	Chamadas de Saída

#1	1	ven:titulo(l,t)	8	a:descricao(l,tl), a:Livro(l);
				a:descricao(l,tl), a:DVD(l);
				a:descricao(l,tl), a:Musica(l)
					Ø	Ø	#2,#5,#6

#2	2	ven:autor(l,au)	10	a:autor(l,au), a:Livro(l)	Ø	Ø	#3

#3	2	ven:preco(l,p)	9	a:preco(l,p), a:Livro(l);
				a:preco(l,p), a:DVD(l);
				a:preco(l,p), a:Musica(l)
					Ø	Ø	#4

#4	2	a:descrição(l,tl)	-	-	Ø	Ø	-
#5	3	ven:autor(l,au)	10(*)	a:autor(l,au), :Livro(l)	Ø	{3}	-
#6	4	ven:autor(l,au)	10(*)	a:autor(l,au), :Livro(l)	Ø	{3,4}	-

Tabela 4.5. Exemplo de execução do Algoritmo *ReescritaSPARQLD* (*Q* do Exemplo 2)

4.5.4 Considerações Relevantes

Nesta seção, abordamos alguns aspectos relevantes no que diz respeito ao algoritmo de reescrita apresentado.

- Nós ilustramos restrições com a utilização do operador "=". Entretanto, observe, no algoritmo *CompatibilizaContexto*, que as linhas 8 e 25 são genéricas o suficiente para lidar com qualquer um dos operadores listados na Seção 3 (=, <, >, =>, <= e .

- A restrição na ontologia fonte (linhas 13 a 34 do algoritmo *CompatibilizaContexto*) não é verificada no filtro, mas sim nos predicados de *built-in* presentes em *pQuery*. Isto acontece porque este tipo de restrição é inserida na consulta sempre através dos mapeamentos. Dessa forma, não haverá valores relativos a elas no filtro da consulta do usuário.

- Quando um conceito que não está presente na ontologia fonte é informado na consulta, o ramo que o contém é eliminado (*ramo falho*), pois tal conceito não pode ser recuperado, uma vez que não existe na fonte.

- Um padrão de tripla do tipo *(?x||A) rdf:type ?y*, onde o sujeito é uma variável *x* ou uma *URI A* e o objeto é uma variável *y*, não necessita ser reescrito, pois tal tipo de padrão não contém informações a respeito do vocabulário das ontologias. O mesmo

acontece para padrões de triplas onde o predicado é uma variável. Por este motivo, não tratamos estes casos em nosso algoritmo.

4.6 Geração da Consulta Local

Com a *árvore SPARQLD de Representação e Derivação* da consulta concluída, assim como o conjunto de atualizações necessárias, é possível gerar a consulta local, que deve ser executada sobre um conjunto de dados. Conforme discutimos nos capítulos anteriores, estes dados podem estar disponíveis em diversos formatos, tais como RDF, relacional ou XML. Para estes dois últimos, a ontologia fonte representa uma tradução sintática do esquema da base de dados. Com isso, uma vez que já existem diversos trabalhos, descritos no Capítulo 2, que realizam a tradução de SPARQL para *XQuery* ou SQL, nós apresentamos, nesta seção, a obtenção da consulta local expressa em SPARQL, visto que nosso foco é a reescrita entre ontologias.

4.6.1 Passos para Geração da Consulta

A consulta local expressa na linguagem SPARQL é construída através da geração de cada um de seus blocos. Para isto, inicialmente, criamos os blocos mais robustos, ou seja, o padrão de grafo (GP) e o formato de retorno. Em seguida, obtemos os demais blocos, correspondentes à cláusula do *Dataset* e aos modificadores de solução. Esclarecemos que não fazemos menção ao prólogo da consulta por este ser um trecho que identifica apenas os prefixos de *URIs*.

Antes de prosseguirmos com a descrição da geração da consulta, uma observação importante diz respeito às seguintes informações, possivelmente configuradas pelo usuário:

- Se os resultados necessitam ser retornados de maneira tabular ou na forma de triplas da ontologia alvo. Como mencionado na etapa de *Normalização da Consulta*, se a opção for pela utilização das triplas, um *CONSTRUCT* padrão será utilizado, caso nenhum *CONSTRUCT* esteja definido.
- Se a consulta deve ser formulada sobre ambas as ontologias fonte e alvo. Observe que o retorno da consulta submetida sobre a ontologia fonte obedece à estrutura e às restrições da ontologia alvo, o que é garantido tanto pelos mapeamentos quanto pela reescrita. Assim, os dados oriundos da fonte podem

ser facilmente unificados com as instâncias presentes no alvo, caso estas existam.

Geração do Padrão de Grafo

O algoritmo exibido na Figura 4.13 é o responsável por esta tarefa. Em suma, ele percorre a árvore e faz uso das anotações e das informações sobre os ramos eliminados para gerar o conteúdo da cláusula *WHERE* da consulta reescrita. É válido ressaltar que se a opção for por uma consulta sobre as ontologias fonte e alvo, uma cláusula *UNION*, contendo o padrão de grafo da consulta original, é adicionada ao *WHERE*.

Geração do Formato de Retorno

Uma vez que a consulta foi corretamente reescrita, é na cláusula referente ao formato de retorno que definimos como os resultados serão estruturados e devolvidos. Nesse sentido, tais resultados podem ser retornados: (i) de forma tabular (SELECT) ou (ii) como um conjunto de triplas na ontologia alvo (CONSTRUCT), de forma que o retorno segue a estrutura desejada para as triplas.

Para os casos (i) e (ii) são usados, respectivamente, os conjuntos R_S e R_C da consulta normalizada Q_N. Além disso, em ambos os casos, o conjunto T (*termos atualizados*), obtido na etapa anterior, também é utilizado, como descrito a seguir.

Seja $T = \{v_1/f_1,..., v_n/f_n\}$, onde cada v_i é uma variável, cada t_i é uma função e $i = 1$ até n. Seja V_T o conjunto das variáveis $\{v_1,..., v_n\}$ de T. Sejam ainda V_S e V_C os conjuntos de variáveis de R_S e R_C, respectivamente.

1. V_S (ou V_C) é disjunto de V_T.

Nesta situação, podemos simplesmente montar a cláusula SELECT (CONSTRUCT) de acordo com R_S (R_C).

2. V_S (ou V_C) não é disjunto de V_T.

Nesta circunstância, temos que uma ou mais informações contidas no retorno da consulta devem ser construídas utilizando as *skolem functions* presentes nos mapeamentos. Por exemplo, para a consulta do Exemplo 4.1, a variável *e*, contida tanto em R_S quanto em R_C, também compõe o conjunto $T = \{e/fEditora(ne)\}$. Isto significa que, durante a construção dos resultados da consulta (instâncias), a variável *e* deve receber o valor correspondente à *fEditora(ne)*. De posse dessa informação, esta atribuição de valor pode ser realizada de duas maneiras principais:

(i) Definindo a função *fEditora* externamente à consulta. Nesta abordagem, as instâncias oriundas do retorno da consulta passam por um processo complementar, utilizando a função *fEditora*. Observe que esta opção permite que funções mais robustas sejam empregadas.

(ii) Inserindo a função diretamente no corpo da consulta. Nesta situação, o processo é menos custoso, mas a definição da função está limitada às capacidades da linguagem.

Neste trabalho, conforme já explanado, uma *skolem function* é uma função que deve retornar como saída um identificador de objeto. Por outro lado, em ontologias, um objeto é unicamente identificado através de uma URI. Portanto, uma possível maneira de se construir estes identificadores de objeto consiste em concatenar a URI da ontologia com o valor da propriedade recebida como entrada, ou seja, para uma *skolem function fSkolem(p)*, seu retorno seria: *fSkolem(p) = URIBaseOntologia#p*, onde *p* é uma propriedade que identifica unicamente um determinado conceito.

Atualmente, em SPARQL, não é possível definir este tipo de operação nas cláusulas SELECT e CONSTRUCT. No entanto, na nova versão da linguagem (SPARQL 1.1), é aceitável implementar *fSkolem* no corpo da consulta, da seguinte forma: *LET(?var := IRI(fn:concat(URIBase,?p))*, onde *?var* é a variável que deve ser substituída pelo retorno de *fSkolem*, e *?p* é o parâmetro de entrada da função. Salientamos que esta geração da URI pode ainda fazer uso de serviços existentes de desambiguação de URIs, entre outros.

Note que a definição da *skolem function* trata-se apenas de uma escolha sobre como implementá-la. O processo de reescrita é genérico o suficiente para permitir que funções de reestruturação quaisquer estejam presentes na cabeça das regras. Além disso, as informações armazenadas no conjunto *T* possibilitam a identificação das funções que devem ser empregadas, assim como as variáveis que receberão o retorno de tais funções. Isto independe da forma como elas estão implementadas (na consulta ou através de operações externas), desde que gerem os identificadores de maneira correta.

Geração dos demais blocos de construção: Cláusula do *Dataset* e Modificadores de Solução

A construção destes dois blocos ocorre de maneira trivial, da seguinte forma:

(i) A cláusula do *Dataset* é composta da palavra-chave "FROM", juntamente com a indicação da ontologia fonte e, opcionalmente, da ontologia alvo.

(ii) Os modificadores de solução permanecem os mesmos, ou seja, seguem o conjunto *SM* da consulta normalizada Q_N. Isto acontece porque tais modificadores não sofrem alterações durante o processo de reescrita.

```
ENTRADA: Árvore SPARQLD noQ
SAÍDA: Padrão de grafo da consulta expresso na linguagem SPARQL
GeraGPSPARQL (no)
1.  Se no não está no conjunto de ramos eliminados Então
2.     Se no.GP = BGP Então geraBGP(no);
3.     Senão geraGGP(no);
4.     Fim Se
5.  Fim Se
Fim GeraGPSPARQL
geraBGP(n)
6.  qRW_BGP := " ";
7.  insereAND:= falso;
8.  Para cada predicado p em noCQ Faça
9.     Se é predicado de built-in Então
10.       Insere no filtro;
11.    Senão
12.       Se insereAND = falso Então
13.          insereAND:= verdadeiro;
14.          qRW_GP := tripla(p);
15.       Senão
16.          qRW_GP :=  qRW_GP + " . " + tripla(p);
17.       Fim Se
18.    Fim Se
19. Fim Para
20. Se n.filter <> vazio e n.filter.raiz não estiver marcada Então
21.    R:= GerarFiltro(filter F);  //Percorre o filtro, eliminado as operações marcadas
22.    qRW_BGP :=  qRW_BGP + "FILTER (" + R + ")";
23. Fim Se
24. Retorne  qRW_BGP;
Fim geraBGP
geraGGP(n)
25. qRW_GGP := " ";
26. insereUnion := falso;
27. Para cada filho n de noQ Faça
28.    Se n não está no conjunto de ramos eliminados Então
29.       Se insereUnion = falso Então
30.          insereUnion = verdadeiro;
31.          qRW_GGP := "{" + GeraGPSPARQL(n) + "}";
32.       Senão
33.          qRW_GGP := n.GP + "{" + GeraGPSPARQL(n) + "}";
34.       Fim Se
35.    Fim Se
36. Fim Para
37. Se n.filter <> vazio Então
38.    R:= GerarFiltro(filter F);  //Percorre o filtro, eliminado as operações marcadas
39.    qRW_GGP :=  qRW_GGP + "FILTER (" + R + ")";
40. Fim Se
41. Retorne qRW_GGP;
Fim geraGGP
```

Figura 4.16. Algoritmo *GeraGPSPARQL*

Finalmente, visando ilustrar a última atividade do processo de reescrita, a Figura 4.16 apresenta a consulta do Exemplo 4.1 reescrita em SPARQL. Além disso, as Figuras 4.17 e 4.18 exibem exemplos de retorno desta consulta tanto de maneira tabular (Figura 4.17) quanto na forma de triplas (Figura 4.18).

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a: <http://amazon.com#>

SELECT ?tl ?au ?e ?ne ?ee ?tv ?dir
FROM <Amazon.owl>
WHERE
{
   {   ?l a:descricao ?tl .
       ?l rdf:type a:Livro .
       ?l a:autor ?au .
       ?l a:preco ?p
       OPTIONAL
       {
         ?l a:enditora ?ne .
         ?l rdf:type a:Livro .
         ?l a:enderecoEditora ?ee}
        }
       FILTER (?p > 30)
   }
   UNION
   {
       ?v rdf:type a:Video .
       ?v a:descricao ?tv .
       ?v a:temDiretor ?_z20 .
       ?_z20 a:nomeDiretor ?dir
   }
   LET (?e := IRI(fn:concat(a:,?ne)))

} ORDER BY ?tl ?tv
```

Figura 4.17. Consulta SPARQL do Exemplo 4.1 reescrita

tl	au	e	ne	ee	tv	dir
“Dom Casmurro”	“Machado de Assis”	<http://amazon.com#Objetiva>	“Objetiva”	“Rua Imperatriz,159”		
					“Avatar”	“J.Cameron”

Figura 4.18. Retorno da consulta de forma tabular

(<http://amazon.com#DC>, ven:titulo, “Dom Casmurro”)
(<http://amazon.com#DC>, ven:autor, “Machado de Assis”)
(<http://amazon.com#DC>, ven:publicadaPor, <http://amazon.com#Objetiva>)
(<http://amazon.com#Objetiva>, ven:nomeEditora, “Objetiva”)
(<http://amazon.com#Objetiva>, ven:enderecoEditora, “Rua Imperatriz,159”)
(<http://amazon.com#Avatar>, ven:titulo, “Avatar”)
(<http://amazon.com#Avatar>, ven:diretor, “J.Cameron”)

Figura 4.19. Retorno da consulta como um conjunto de triplas

4.7 Conclusões

Neste capítulo, apresentamos, em detalhes, um processo para reescrita de consultas entre ontologias com estruturas distintas, o qual faz uso de mapeamentos heterogêneos e noções de programação em lógica. Para descrição do processo proposto, expomos as principais características de cada uma de suas atividades: (i) Normalização da Consulta, (ii) Conversão de SPARQL para Regras, (iii) Reescrita da Consulta e Reestruturação dos Resultados e (iv) Geração da Consulta Local. Mais especificamente, em relação à atividade (iii), que consiste no núcleo de processo de reescrita, exibimos um algoritmo que: (i) é capaz de lidar com os mapeamentos heterogêneos; (ii) manipula as informações de contexto presentes nas regras de mapeamento e (iii) possibilita a eliminação de trechos desnecessários da consulta reescrita gerada. Por fim, apresentamos um exemplo de execução deste algoritmo, discutimos alguns aspectos relevantes da solução adotada e mostramos como uma consulta local pode ser gerada na linguagem SPARQL.

CAPÍTULO 5
Aspectos de Implementação

Neste capítulo, apresentamos os aspectos de implementação relacionados ao processo proposto no capítulo anterior. Para isto, na Seção 5.1, descrevemos a ferramenta desenvolvida, ilustrando suas principais características e funcionalidades. A Seção 5.2, por sua vez, aborda aspectos referentes à validação do trabalho. Para tanto, inicialmente, na Seção 5.2.1, levantamos uma discussão a respeito dos critérios adotados para realizarmos as medições e as comparações. Em seguida, na Seção 5.2.2, discorremos sobre o cenário de avaliação construído. Nas Seções 5.2.3 e 5.2.4, exibimos os resultados obtidos, juntamente com uma análise a respeito destes resultados. Por fim, a Seção 5.3 apresenta a conclusão.

5.1 A Ferramenta *SQuOL*

Visando implementar e testar o processo apresentado no capítulo anterior, desenvolvemos a ferramenta SQuOL (*SPARQL Query Rewriter between Ontologies using Heterogeneous Mappings and Logic Programming*). Esta ferramenta permite que o usuário, através de uma interface gráfica amigável, formule e reescreva consultas, manipule mapeamentos, entre outras funcionalidades que serão detalhadas mais adiante.

5.1.1 Arquitetura

A Figura 5.1 exibe uma visão geral e resumida da arquitetura da ferramenta SQuOL, seguida de uma breve explanação sobre seus módulos.

- GUI (*Graphical User Interface*): Módulo responsável pela interação entre o usuário e as funcionalidades do sistema. Tais funcionalidades estão vinculadas à definição de preferências, formulação das consultas, visualização de ontologias, consultas e resultados. Além disso, este módulo se comunica com o módulo de *Armazenamento e Manipulação dos Mapeamentos*, com o intuito de disponibilizar os mapeamentos para que o usuário possa escolhê-los, editá-los, excluí-los ou alterá-los.
- Módulo de Armazenamento e Manipulação dos Mapeamentos: Armazena os mapeamentos em documentos XML definidos de acordo com o esquema apresentado no

Apêndice C. A opção pela utilização de XML deve-se ao fato desta ser uma linguagem simples, flexível e de grande utilização para intercâmbio de informações na *Web*. Além disso, linguagens existentes para representação de regras, tais como RULE ML e SWRL, tem representações construídas como extensões da linguagem XML. Dessa forma, pudemos nos basear nestas linguagens para retratarmos nossas regras de mapeamento. A Figura 5.2 ilustra um exemplo com a descrição de duas regras.

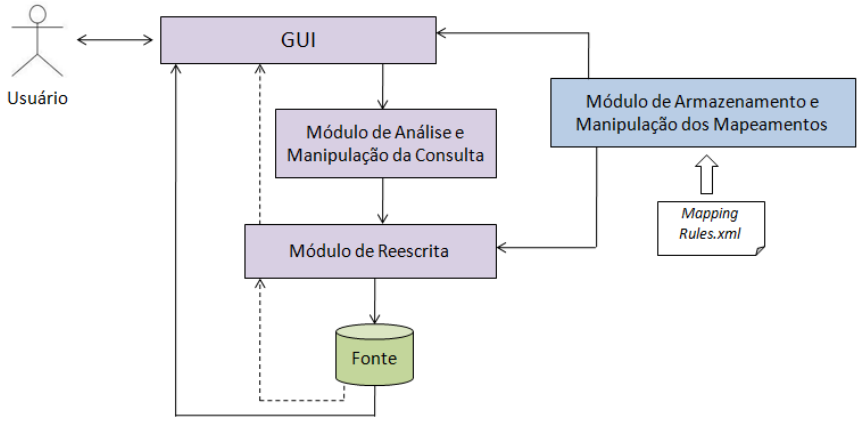


Figura 5.1. Arquitetura geral da ferramenta SQuOL

- Módulo de Análise e Manipulação da Consulta: Responsável por executar a primeira e a segunda atividade do processo descrito no capítulo anterior, ou seja: *Normalização da Consulta* e *Conversão de SPARQL para Regras*.

- Módulo de Reescrita: Reflete a atividade de reescrita, com seus respectivos algoritmos, bem como gera a consulta local. Vale ressaltar que, caso a função de reestruturação não esteja definida diretamente na consulta, o resultado obtido pode ainda passar por um tratamento adicional neste módulo, a fim de que tal resultado possa ser retornado de acordo com a ontologia alvo.

A arquitetura exibida foi implementada na linguagem JAVA. A API Jena[11] foi utilizada tanto para manipular as ontologias quanto para lidar com as consultas em SPARQL, juntamente com a API ARQ[12]. Para trabalhar com os arquivos XML, utilizamos a JDOM[13].

5.1.2 Funcionalidades

A Figura 5.3 apresenta o diagrama de casos de uso correspondente às funcionalidades da SQuOL.

[11] http://jena.sourceforge.net/

[12] http://jena.sourceforge.net/ARQ/app_api.html

[13] http://www.jdom.org/

r1: v:Musica(p) ← e:Produto(p), e:tipo(p,k), k= “musica”
r12: v:distribuidaPor(m,fGravadora(y)) ← a:interpretadaPor(m,z),a:gravadora(z,y), a:Cantor(z)

```
<mappingRule id="r1">
  <head>
    <classAtom class="v:Musica">
      <args>
        <var>p</var>
      </args>
    </classAtom>
  </head>
  <body>
    <conjunction>
      <classAtom class="e:Produto">
        <args>
          <var>p</var>
        </args>
      </classAtom>
      <context>
        <contextSource>
          <restrictionSource>
            <propertyAtom property="e:tipo">
              <args>
                <var>p</var>
                <var>k</var>
              </args>
            </propertyAtom>
            <builtinAtom builtin="swrlb:equal">
              <args>
                <var>k</var>
                <constant dataType="xsd:string">musica</constant>
              </args>
            </builtinAtom>
          </restrictionSource>
        </contextSource>
      </context>
    <conjunction>
  </body>
</mappingRule>
```

```
<mappingRule  id="r12">
  <head>
    <propertyAtom property="v:distribuidaPor">
      <args>
        <var>m</var>
        <function name="fGravadora">
          <args>
            <var>y</var>
          </args>
        </function>
      </args>
    </propertyAtom>
  </head>
  <body>
    <conjunction>
      <propertyAtom property="a:interpretadaPor">
        <args>
          <var>m</var>
          <var>z</var>
        </args>
      </propertyAtom>
      <propertyAtom property="a:gravadora">
        <args>
          <var>z</var>
          <var>y</var>
        </args>
      </propertyAtom>
      <context>
        <contextSource>
          <classAtom class="a:Cantor">
            <args>
              <var>z</var>
            </args>
          </classAtom>
        </contextSource>
      </context>
    <conjunction>
  </body>
</mappingRule>
```

Figura 5.2. Exemplo de regras de mapeamento em um documento XML

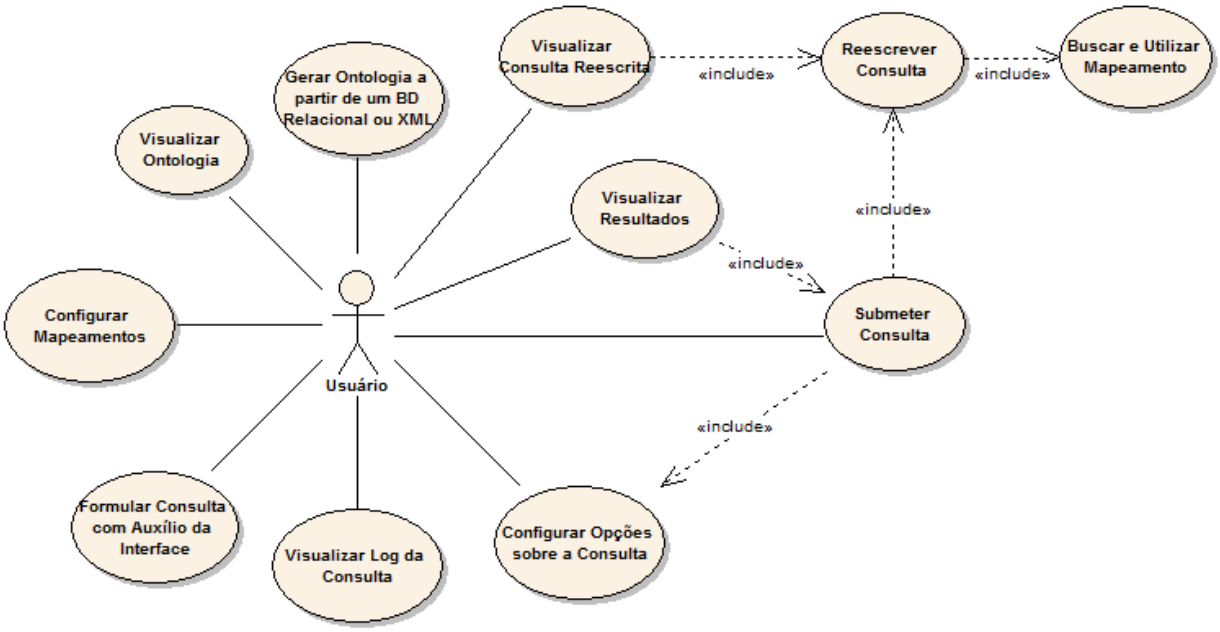


Figura 5.3. Funcionalidades da SQuOL

A respeito destas funcionalidades, detalharemos, a seguir, apenas duas delas, uma vez que o entendimento das demais é bastante intuitivo.

- Configurar Mapeamentos: permite que o usuário visualize, selecione, edite, adicione e remova um mapeamento. A edição pode ser realizada tanto sobre o documento XML criado quanto sobre as regras definidas em uma extensão de *Datalog*. Neste último caso, tais alterações são refletidas no XML.

- Configurar Opções sobre a Consulta: o usuário determina se a consulta deve ser executada sobre ambas as ontologias ou somente sobre a ontologia fonte. Além disso, indica como deseja receber os resultados: de maneira tabular (projeção) ou na forma de triplas da ontologia alvo.

A Figura 5.4 exibe a tela principal da ferramenta, na qual é possível visualizar cinco áreas:

(a) Área da ontologia alvo (*Target Ontology*): permite a visualização da ontologia alvo, com suas respectivas classes e propriedades.

(b) Área da consulta original (*Original Query*): local onde é inserida a consulta que deve ser reescrita.

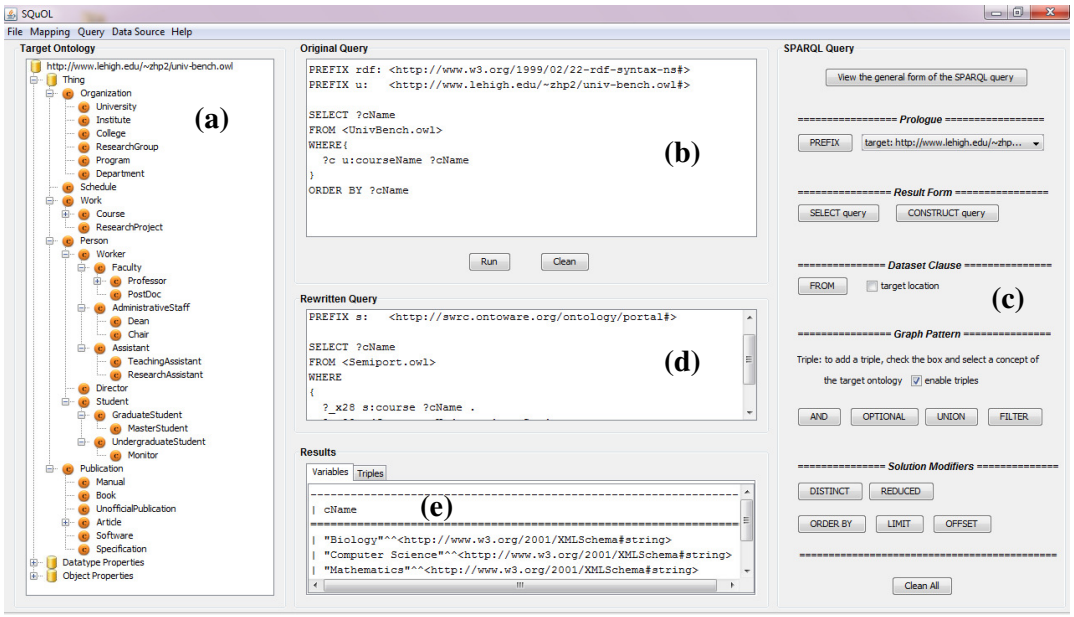


Figura 5.4. Tela Principal da Ferramenta *SQuOL*

(c) Área de auxílio à formulação de consultas em SPARQL (*SPARQL Query*): ajuda o usuário na formulação das consultas. Para isto, são disponibilizados *wizards* para cada um dos blocos de construção da linguagem SPARQL. Assim, à medida que a consulta é construída, ela vai sendo inserida na área (b). Se o usuário desejar, ele pode

ainda clicar sobre os elementos da ontologia contida na área (a), de maneira que estes elementos são adicionados (na forma de conceitos ou como padrões de triplas) na área (b).

(d) Área da consulta reescrita (*Rewritten Query*): apresenta a consulta reescrita.

(e) Área de exibição dos resultados: expõe os resultados da consulta, caso esta seja submetida. Tais resultados podem ser retornados de maneira tabular (aba *Variables*) ou na forma de triplas da ontologia alvo (aba *Triples*).

Já a Figura 5.5 apresenta a tela referente à manipulação dos mapeamentos, a qual possui as seguintes áreas:

(a) Lista dos mapeamentos contidos no repositório (*Mappings*): possibilita a visualização, edição e remoção dos arquivos de mapeamento contidos no repositório. Além disso, permite a inserção de novos mapeamentos.

(b) Arquivo das Regras de Mapeamento (*File*): exibe o arquivo XML do mapeamento selecionado na área (a).

(c) Regras do mapeamento selecionado (*Rules*): expõe o conjunto de regras contidas no mapeamento selecionado em (a). Nesta área, é possível adicionar, remover ou editar tais regras.

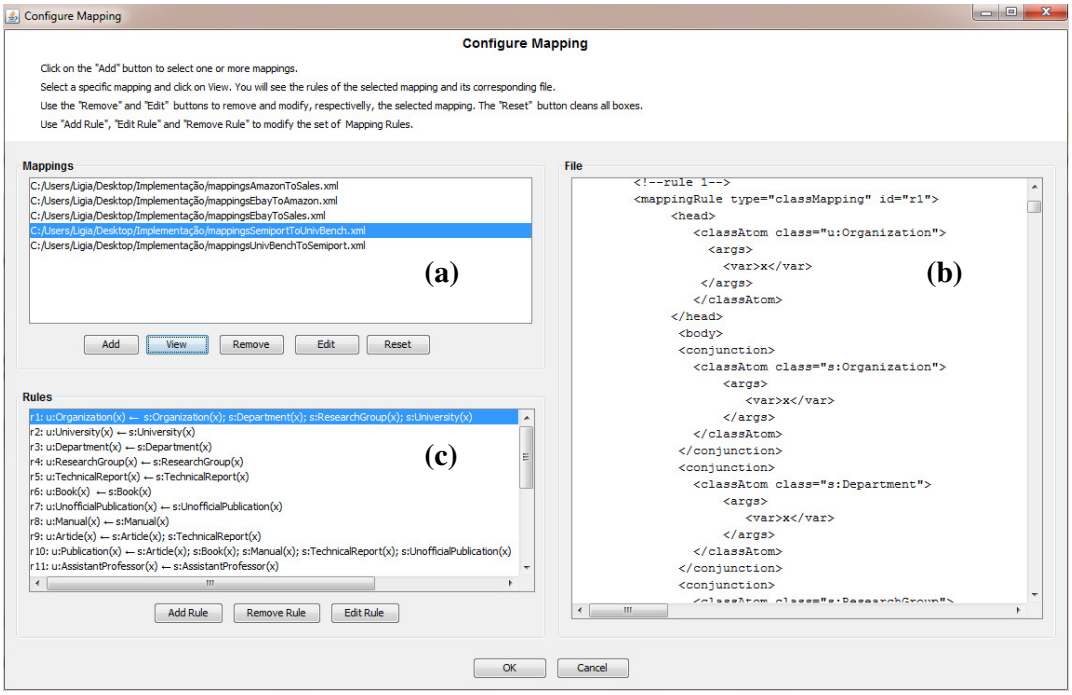


Figura 5.5. Tela para manipulação dos mapeamentos

5.2 Validação Experimental

A validação experimental apresentada nesta seção tem como objetivo avaliar, na prática, o processo de reescrita proposto, em termos de sua corretude e completude, assim como compará-lo com outras abordagens que não levam em consideração os aspectos tratados neste trabalho. Para tanto, inicialmente, apresentamos uma discussão a respeito dos critérios adotados para realizarmos as medições e as comparações. Em seguida, discorremos sobre o cenário de avaliação construído. Por fim, exibimos os resultados obtidos, juntamente com uma análise a respeito destes resultados.

5.2.1 Objetivos e Critérios para Medições e Comparações

Corretude e Completude

Para mostrarmos, em termos práticos, o grau de corretude e completude do método de reescrita, é necessário que tenhamos algum parâmetro de referência que nos indique quais deveriam ser os resultados obtidos. A partir de então, poderíamos comparar os nossos resultados com estes, de forma a verificarmos se eles estão de acordo.

No contexto do problema de reescrita tratado, para definirmos esta referência, podemos utilizar a premissa de que os mapeamentos recebidos como entrada são corretos, uma vez que o objetivo não é avaliar a corretude destes mapeamentos, mas sim do processo de reescrita. Isto significa que, de posse de (i) duas ontologias O_S e O_T e de (ii) um conjunto de mapeamentos M_{ST} entre elas, se utilizarmos estes mapeamentos para transferirmos todas as instâncias de O_S para O_T, estas novas instâncias contidas em O_T estarão coerentes com os dados em O_S. Dessa forma, com todos os dados em O_T, podemos submeter as consultas diretamente sobre a ontologia alvo, sem a necessidade de reescrita.

Por outro lado, não estamos interessados em povoar O_T com todas as instâncias provenientes de O_S. Nosso intuito é utilizar estes mesmos mapeamentos M_{ST} para reescrevermos as consultas submetidas sobre O_T, de forma a recuperarmos somente as instâncias que nos interessam em O_S. Observe que, como os mapeamentos são os mesmos, podemos avaliar se nosso processo de reescrita está funcionando de maneira adequada, da seguinte forma:

- Seja P_T um processo que povoa, de uma só vez, a ontologia alvo O_T com todas as instâncias da ontologia fonte O_S, utilizando M_{ST}. Chamaremos esta ontologia de O_{TI} (observe que, para este teste, O_{TI} possui apenas instâncias provenientes de O_S).

- Seja P_R o processo que reescreve uma consulta Q, submetida sobre O_T, para uma consulta Q' sobre O_S.

- Seja R_T o retorno obtido com a submissão de Q sobre O_{TI}.

- Seja R_W o retorno obtido com a submissão de Q' sobre O_S.

- Se $R_T = R_W$, então o processo de reescrita é correto e completo, pois ele recupera todos os resultados corretos que deveria obter.

Diante do exposto, para compararmos nossos resultados com os resultados esperados, realizamos os passos descritos abaixo, os quais estão ilustrados na Figura 5.6.

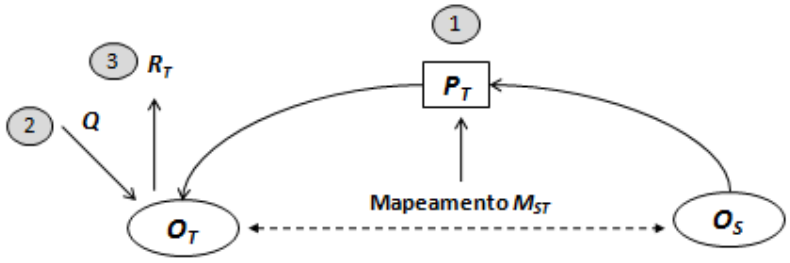


Figura 5.6. Passos para geração do conjunto de referência dos valores de retorno da consulta

1. Criamos um *script* que representa o processo P_T. Neste *script*, para cada regra de mapeamento, povoamos O_T com as instâncias de O_S.
2. Com O_T totalmente povoada, submetemos uma consulta Q diretamente sobre tal ontologia.
3. Como resultado desta consulta, é gerado um conjunto de instâncias R_T. Este é o conjunto utilizado como referência para cada umas das consultas Q' originadas a partir da reescrita de Q.

Uma vez determinado o conjunto de referência R_T, as seguintes variáveis (baseadas nas métricas *precision* e *recall*) podem ser utilizadas para medir o grau de corretude e completude do processo:

$$Corretude = \frac{\#número\ de\ instâncias\ corretas\ recuperadas\ com\ Q'}{\#número\ total\ de\ instânicas\ recuperadas\ com\ Q'}$$

$$Completude = \frac{\#número\ de\ instâncias\ corretas\ recuperadas\ com\ Q'}{\#número\ total\ de\ instânicas\ em\ RT}$$

É válido salientar que estamos realizando estas medições em um ambiente controlado e supervisionado, visando avaliar o processo proposto. Em um ambiente não supervisionado, nem sempre há como saber quais seriam os resultados ideais, por vários motivos. Um destes motivos é a impossibilidade de transferir todas as instâncias de um esquema fonte para um esquema alvo. Esta é uma das razões, dentre outras, que leva à

adoção da reescrita de consultas.

Comparação com outras abordagens

Para esta comparação, devemos levar em conta algumas particularidades do nosso método que não são analisadas em outras soluções, ou seja: (i) a manipulação de mapeamentos heterogêneos, expressos através de regras, as quais podem possuir funções; (ii) a possibilidade destes mapeamentos possuírem informações de contexto (restrições) tanto na ontologia fonte quanto na ontologia alvo; (iii) o descarte de partes da consulta que seriam redundantes ou retornariam vazio.

Dos trabalhos apresentados na Seção 2.3 (Capítulo 2), os únicos que disponibilizam ferramentas, que podem ser utilizadas para verificações, são os de Calvanese et. al [Poggi et al. 2008; Calvanese et al. 2009] e Fernandes [Fernandes 2009]. No entanto, no trabalho de Calvanese, as consultas reescritas são submetidas diretamente sobre um SGBD relacional. Por outro lado, o trabalho de Fernandes admite apenas consultas envolvendo classes, além de ter um foco distinto do nosso, pois a ideia principal da autora consiste em enriquecer as consultas. Além disso, nenhum destes dois trabalhos permite a inserção de uma consulta *SPARQL* livre.

Com isso, para termos como confrontar a nossa abordagem, pelo menos em parte, com as demais, nós consideramos dois tipos de consulta:

1. Q_{SQU}: Dado uma consulta Q, Q_{SQU} é a consulta reescrita gerada através do processo implementado pela ferramenta SQuOL.
2. Q_{BAS}: Dado uma consulta Q, Q_{BAS} é a consulta que seria originada por uma abordagem básica, que não considerasse a verificação do contexto nas regras, funções e descartes.

Destacamos que a Q_{BAS} também utiliza os mapeamentos M_{ST}, de forma que estes dois tipos de consulta podem se assemelhar, em algumas situações. Argumentamos que, por hora, não há meios de compararmos diretamente a implementação de nossa abordagem com as existentes, pois além destas tratarem aspectos distintos da reescrita, a maioria delas, como já mencionado, não disponibiliza protótipos.

5.2.2 Cenário de Avaliação

A avaliação experimental foi desenvolvida em uma máquina HP *Pavilion* com *Linux Ubuntu* 8.04, processador Intel Core TM 2 Duo e 3GB de memória. Para esta avaliação, montamos um cenário que envolve:

- 5 ontologias[14] divididas em dois grupos, relativos aos domínios de Vendas e Educação. As ontologias do primeiro grupo (*Sales.owl*, *Amazon.owl* e *Ebay.owl*) são baseadas nas adotadas por [Sacramento et. al 2010] e [Leme 2009]. Já as do segundo grupo (*UnivBench.owl* e *Semiport.owl*), são ontologias que se encontram disponíveis na internet para *download*, as quais também foram utilizadas no sistema apresentado em [Pires 2009]. Trechos relevantes destas ontologias podem ser vistas no Apêndice C.
- Um conjunto de mapeamentos[15], gerados conforme descrito no Capítulo 3, entre estas ontologias. Em cada um destes mapeamentos, as ontologias fonte e alvo são, respectivamente: (i) *Amazon.owl* e *Vendas.owl*; (ii) *Ebay.owl* e *Vendas.owl*; (iii) *Amazon.owl* e *Ebay.owl*; (iv) *Semiport.owl* e *UnivBench.owl*; (v) *UnivBench.owl* e *Semiport.owl*. As regras correspondentes a estes mapeamentos podem ser vistas no Apêndice E.
- Um conjunto com 26 (vinte e seis) consultas que abrangem: (i) os principais construtores da linguagem SPARQL; (ii) a necessidade de utilização de cada um dos tipos (casos) de mapeamentos descritos no Capítulo 3; (iv) semânticas variadas, como por exemplo, consultas conjuntivas, disjuntivas e com negação (em SPARQL, a negação é possível através da *negação por falha*). O Apêndice F contém estas consultas, apresentando-as da seguinte forma: (i) consulta original em linguagem natural e em SPARQL; (ii) consultas reescritas Q_{SQU} e Q_{BAS}.

5.2.3 Experimentos e Resultados dos Experimentos

Uma vez definido o cenário, dois tipos de experimentos, explanados adiante, foram realizados, ambos considerando as consultas Q_{SQU} e Q_{BAS}.

Experimento 1

O objetivo deste experimento é ilustrar o tipo de consulta que cada uma das abordagens é capaz de reescrever, bem como averiguar aspectos referentes à corretude e à completude das consultas reescritas. Para tanto, os passos descritos na Seção 5.2.1 foram realizado, com o intuito de determinar, para cada consulta de entrada, o resultado de referência a ser utilizado para as comparações.

[14] [5] Disponíveis para download em: www.lia.ufc.br/~fernanda.ligia/SQUOL

A Tabela 5.1 ilustra os resultados obtidos utilizando o processo implementado pela ferramenta *SQuOL (*PQ_{SQU}*)* e um processo que geraria as consultas reescritas do tipo Q_{BAS} (PQ_{BAS}).

Tabela 5.1. Resumo com os resultados sobre as consultas obtidas por PQ_{SQU} e PQ_{BAS}

Conjunto de Consultas	PQ_{SQU}	PQ_{BAS}
CNR	0	Q5, Q6, Q10, Q20, Q21, Q26
CRV	0	Q3, Q12, Q13
CRI	0	Q4
CRC	Q1-Q26	DEMAIS CONSULTAS

Legenda:

CNR: Consultas Não Reescritas
CRV: Consultas Reescritas que retornaram Vazio, quando deveriam retornar resultados
CRI : Consultas com retorno diferente de vazio, mas com alguns Resultados Incorretos
CRC: Consultas com o Retorno Correto (de acordo com o conjunto de referência)

A respeito destes resultados, é possível visualizar que, para todas as consultas testadas, o método de reescrita proposto recuperou todas as instâncias que deveria (*completude*), além de não originar respostas erradas (*corretude*). Por outro lado, o método PQ_{BAS} apresentou alguns resultados indesejados, conforme discutido a seguir.

As consultas inseridas em CNR são aquelas que utilizam mapeamentos mais complexos, tais como os com funções na cabeça das regras. Por este motivo, PQ_{BAS} não é capaz de respondê-las.

Já as contidas em CRV, são consultas para as quais PQ_{BAS} realiza uma rescrita. No entanto, tal consulta não retorna resposta alguma, ao passo que o conjunto de referência não é vazio. Estas consultas condizem com as que fazem uso de mapeamentos que refletem restrições na ontologia alvo, ou seja, na cabeça das regras de mapeamento.

A consulta Q4, presente em CRI, também faz uso de mapeamentos deste último tipo citado. Entretanto, esta consulta não possui um retorno vazio, mas sim com mais instâncias que o conjunto de referência. Para o experimento realizado, esta consulta teve um grau de *corretude* correspondente a 67%, ou seja, para cada 100 instâncias recuperadas, 67 eram corretas. Além disso, estas instâncias corretas correspondiam ao conjunto que ela deveria recuperar, de fato. Isto significa que o grau de completude foi

de 100%. Nós observamos que a particularidade desta consulta Q4 diz respeito ao fato dela remeter a uma negação por falha em *SPARQL*.

Finalmente, as consultas pertencentes ao CRE são aquelas que fazem indagações mais elementares ou que podem ser respondidas com mapeamentos mais simples, isto é, mapeamentos entre trechos de ontologias com estruturas semelhantes.

Para as consultas consideradas neste Experimento 1, PQ_{SQU} apresentou ambos os graus de corretude e completude iguais a 100%. Já PQ_{BAS} obteve graus de corretude e completude iguais a 61,5% e 65,38%, respectivamente.

A conclusão que podemos tirar deste experimento é que, quanto mais elaborados forem os mapeamentos e as consultas, menos confiável é um processo como o PQ_{BAS}, ao passo que o PQ_{SQU} continua funcionando bem. É válido lembrar que, conforme discutido no Capítulo 3, mapeamentos mais complexos refletem ontologias mais heterogêneas. Finalmente, destacamos o fato de que PQ_{BAS} não é capaz de lidar com funções nas regras de mapeamento. Dessa forma, ele não tem condições de gerar os resultados conforme a ontologia alvo, se necessário.

Experimento 2

O objetivo deste experimento é medir o tempo de execução das consultas reescritas Q_{SQU} e Q_{BAS}, com intuito de verificar como a eliminação de redundâncias e subconsultas de retorno vazio influenciam no tempo de resposta.

Para este experimento, cada uma das 5 ontologias utilizadas foi povoada com 40.000 triplas. A Tabela 5.2 e a Figura 5.7 exibem os dados e o gráfico, respectivamente, para os tempos das 13 primeiras consultas, as quais foram executadas sobre as ontologias do domínio de *Vendas*. Já a Tabela 5.3 e a Figura 5.8 mostram os dados e o gráfico, respectivamente, para as consultas de Q14 a Q26, realizadas sobre as ontologias do domínio de *Educação*.

Nas tabelas, as colunas coloridas simbolizam:

- Verdes: consultas Q_{BAS} que possuem um retorno vazio (CRV), conforme explanado no experimento anterior.
- Azuis: consultas que o PQ_{BAS} não consegue reescrever (CNR).
- Amarela: consulta que o PQ_{SQU} descartou desde a raiz da árvore e, por este motivo, não necessitou ser executada.

A partir dos resultados obtidos, é possível observar que a maioria das consultas Q_{SQU} foi executada em tempo menor que as Q_{BAS}. Isto indica que os descartes realizados

tiveram impacto sobre o tempo de resposta destas consultas. Em Q8, por exemplo, a consulta solicitava o *titulo e o diretor de Vídeos*. Como *título* é uma propriedade pertencente à classe pai *Produto*, ele mapeia no contexto de outras classes, como *Livro* ou *Música*. Entretanto, somente os *Vídeos* possuem *diretor*. Sendo assim, as classes desnecessárias são eliminadas da consulta. Em Q13, são excluídas ainda informações contidas nos filtros.

Por outro lado, as consultas com tempos de execução iguais (Q1, por exemplo) são aquelas para as quais PQ_{SQU} e PQ_{BAS} geram a mesma saída na reescrita, ou seja, se a consulta for simples, PQ_{SQU} mantém a simplicidade da consulta de saída.

De modo geral, na Tabela 5.2 (gráfico da Figura 5.7), um fato que chama a atenção é o tempo de resposta da consulta Q6.

Tabela 5.2. Tempos de execução das consultas SPARQL reescritas por PQ_{SQU} e PQ_{BAS} (ontologias do domínio de Vendas)

Consulta em SPARQL	$Q_{SQU}(ms)$	$Q_{BAS}(ms)$
Q1	92,16	92,16
Q2	88,11	88,11
Q3	94,5	98,1
Q4	113,2	117,21
Q5	98,1	Não Reescreve
Q6	770,4	Não Reescreve
Q7	111,55	159,7
Q8	116,9	177,85
Q9	105,63	150,92
Q10	114,2	Não Reescreve
Q11	119,45	119,45
Q12	95,1	138,09
Q13	102,81	140,72

Acreditamos que este tempo, possivelmente, deve-se ao fato desta ser uma consulta que solicita várias propriedades, pertencentes a classes distintas, além de fazer uso de restrições no filtro (propriedade de contexto *tipo*). Dada a particularidade do tempo de resposta desta consulta, nós realizamos o seguinte teste com ela: uma vez que uma abordagem como PQ_{BAS} não consegue respondê-la, nós ignoramos as funções presentes na cabeça das regras de mapeamento que esta consulta utiliza, bem como não eliminamos restrições ou realizamos quaisquer podas na arvora da consulta. Como consequência, obtivemos uma consulta reescrita que demorou na casa de *minutos* para retornar vazio.

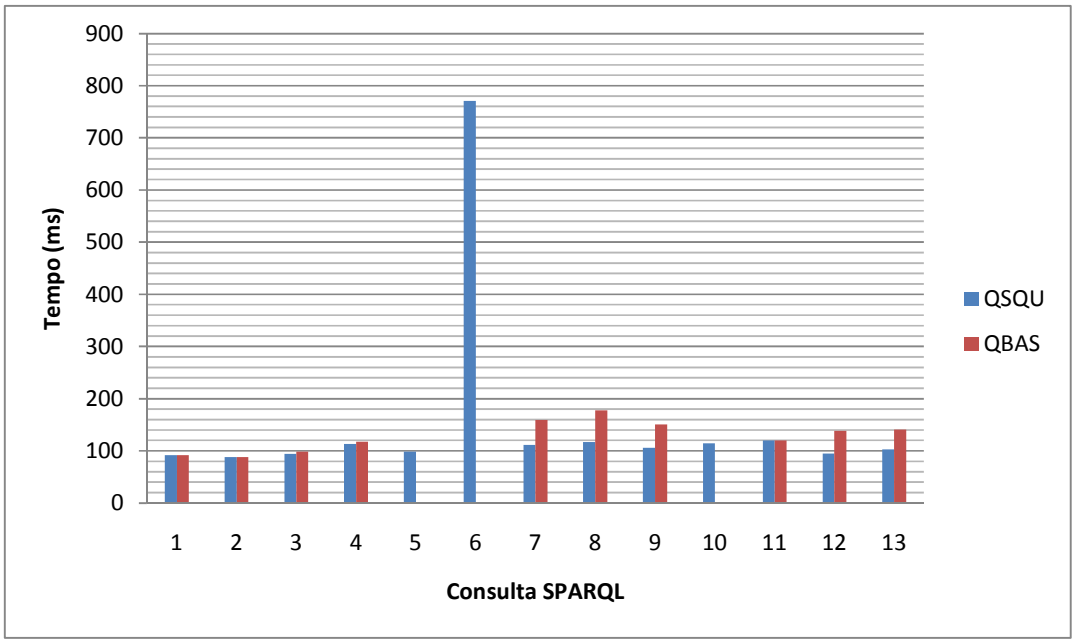


Figura 5.7. Gráficos com os tempos de resposta das consultas SPARQL reescritas por PQ_{SQU} e PQ_{BAS} (ontologias do domínio de Vendas)

Na Tabela 5.3 e no gráfico da Figura 5.8, apontamos como pontos principais as consultas Q14, Q24 e Q25.

Tabela 5.3. Tempos de execução das treze últimas consultas SPARQL reescritas por PQ_{SQU} e PQ_{BAS} (ontologias do domínio de Educação)

Consulta em SPARQL	Q_{SQU}(ms)	Q_{BAS}(ms)
Q14	100,34	180,28
Q15	112,75	112,75
Q16	103,67	121,72
Q17	145,77	188,03
Q18	132,26	132,26
Q19	189,47	195,97
Q20	108,92	Não Reescreve
Q21	114,36	Não Reescreve
Q22	105,46	105,46
Q23	129,89	129,89
Q24	Não Submete	149,43
Q25	106,83	151,66
26	110,8	Não Reescreve

Em Q14, é solicitado o nome de todos que fazem algum curso, juntamente com este curso. Como a propriedade *cursa* (*takesCourse*) se aplica somente a *Estudantes*, então somente estes são necessários na consulta. Já a consulta Q24, pergunta o nome de todos os professores assistentes que orientam alunos de pós-graduação. Na ontologia

alvo, a propriedade *orienta* se aplica a qualquer professor, portanto, esta pergunta é coerente quando aplicada a tal ontologia. Entretanto, na ontologia fonte, o domínio da propriedade *orienta* é a classe correspondente a professores titulares (*Full Professor*), o que determina o contexto da regra de mapeamento como sendo *Full Professor*. Dessa forma, a consulta Q24, quando reescrita e executada, retorna vazio. Como trata-se de uma conjunção (lembre-se que o operador OPTIONAL está dento da conjunção), esta consulta é descartada desde a raiz, não sendo necessário executá-la.

Em Q25, ocorre uma situação semelhante à Q24. No entanto, uma vez somente a parte OPCIONAL retornará vazio, apenas tal trecho é excluído da consulta reescrita.

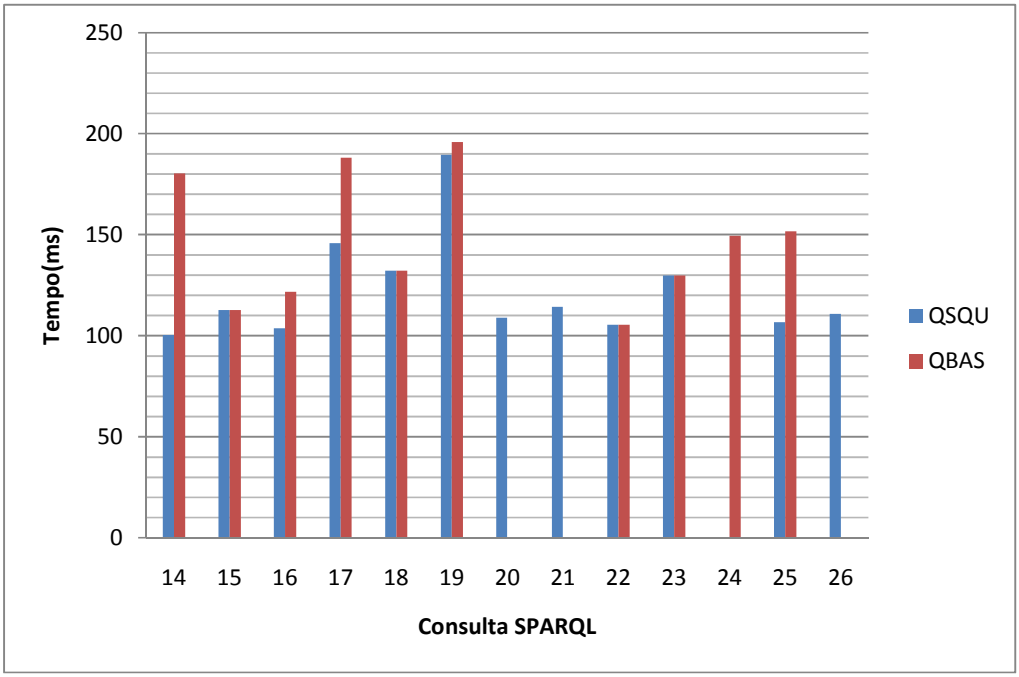


Figura 5.8. Gráficos com os tempos de resposta das consultas SPARQL reescritas por PQ_{SQU} e PQ_{BAS} (ontologias do domínio de Educação)

Finalmente, como conclusão deste experimento, podemos afirmar que é possível diminuir o tempo de execução das consultas reescritas, através da eliminação de informações desnecessária ou que tornem tais consultas incorretas. Para isto, fazemos uso dos contextos contidos nas regras.

5.2.4 Algumas Considerações e Conclusões sobre os Experimentos

Após a realização dos experimentos, é importante tirarmos algumas conclusões e levantarmos alguns pontos.

(i) Conforme discutimos, o que estamos chamando de *processo básico* é, na verdade, um processo semelhante ao nosso, desconsiderando determinados aspectos. No entanto, o tipo de mapeamento utilizado, por exemplo, é o mesmo. Portanto, observe que este *processo básico* é um subconjunto do nosso processo que consegue tratar apenas consultas envolvendo mapeamentos simples.

(ii) No Experimento 1, nós mostramos algumas consultas, que envolvem mapeamentos entre ontologias heterogêneas, que somos capazes de responder, ao passo que outros processos simples não seriam. Por outro lado, exibimos também consultas que são respondidas por ambos. Note que, no Experimento 2, algumas destas consultas respondidas foram reduzidas utilizando a SQ_UOL.

(iii) É importante deixar claro que nosso foco, neste trabalho, não consiste em tratar aspectos de otimização de consultas em SPARQL. A ideia é mostrar que podemos eliminar trechos desnecessários da consulta reescrita, gerando, assim, consultas com uma melhor performance. Determinar se a consulta obtida é a melhor, ou a mais otimizada, não é o objetivo deste trabalho.

(iv) Uma vez que os resultados nos indicaram caminhos positivos, uma análise posterior consiste em montar ou adotar um *banchmark* para aperfeiçoar este tipo de teste, através da utilização de um conjunto massivo de dados.

5.3 Conclusões

Este capítulo apresentou a implementação e a validação do processo de reescrita proposto nesta dissertação. Inicialmente, descrevemos a ferramenta desenvolvida (*SQuOL)*, a qual oferece diversas funcionalidades para que usuários e aplicações lidem com consultas e mapeamentos. Em seguida, apresentamos a validação do trabalho. Para isto, descrevemos os critérios de medições adotados e o cenário no qual os experimentos foram executados.

O primeiro experimento realizado teve como objetivo ilustrar o tipo de consulta que nosso processo é capaz de reescrever, bem como averiguar aspectos referentes à corretude e à completude das consultas rescritas. Para o conjunto de consultas SPARQL considerado, nosso processo reescreveu de maneira correta todas as consultas testadas, mostrando que é capaz de manipular adequadamente os mapeamentos heterogêneos. Já no Experimento 2, comprovamos que a eliminação de trechos desnecessários das consultas reescritas permite a geração de consultas com um menor tempo de resposta.

CAPÍTULO 6
Conclusões

Neste trabalho, apresentamos uma solução para o problema de OBDA (*OntologyBased Data Acess*), através da especificação de um ambiente onde cada uma das atividades vinculadas a este problema é tratada de maneira independente. Com isso, dada esta modularização, nos concentramos no subproblema referente à reescrita de consultas entre ontologias com estruturas distintas, passando abordá-lo de uma maneira mais geral, tendo o cenário de OBDA apenas como uma de suas aplicações.

Em especial, o fato das ontologias apresentarem estruturas distintas trouxe à tona a necessidade de desenvolvermos um processo de reescrita que fosse capaz de lidar com mapeamentos heterogêneos, os quais foram expressos por meio de regras. No processo de reescrita proposto, manipulamos estas regras de mapeamento utilizando noções de Programação em Lógica. Dessa forma, alguns dos conceitos já estabelecidos nesta área nos foram úteis, sendo realizadas as devidas adaptações para o nosso contexto.

Dentre os principais diferenciais da nossa abordagem, destacamos o fato de lidarmos com o problema de reescrita de consultas entre ontologias que apresentam estruturas distintas. Além disso, os mapeamentos heterogêneos utilizados neste trabalho permitem lidar com restrições em ambos os esquemas fonte e alvo. Estas restrições são expressas por meio de predicados de comparação. Na abordagem proposta, consideramos que estes predicados de comparação podem estar presentes tanto nos mapeamentos quanto nas consultas, sendo estes analisados durante o processo de reescrita. Também vale a pena ressaltar que a maioria dos trabalhos relacionados não considera a existência de funções nos mapeamentos, enquanto que nós utilizamos funções para tratar algumas diferenças estruturais. Por outro lado, durante o nosso processo, ainda possibilitamos que partes da consulta possam ser descartadas, caso seja constatado que tais partes retornariam vazio. Por fim, ainda mencionamos o fato de que, em nossa estratégia, os resultados podem ser retornados conforme a ontologia alvo, se isto for de interesse da aplicação. Dentre os trabalhos relacionados analisados, nenhum trata todos estes aspectos.

É importante destacar que, para a validação da abordagem proposta, implementamos um protótipo e realizamos experimentos considerando diferentes ontologias e diversos tipos de consulta, as quais abrangem (i) a maioria dos construtores

da linguagem SPARQL e (ii) os casos de heterogeneidade apresentados no Capítulo 3. Como resultado desta avaliação, verificamos que as consultas foram reescritas de maneira correta pelo nosso processo. Além disso, mostramos que a eliminação, realizada por nossa estratégia, de trechos desnecessários das consultas reescritas, permite a geração de consultas com um menor tempo de resposta.

6.1 Trabalhos Futuros

A seguir, apresentamos algumas direções que podem ser exploradas em trabalhos futuros:

- Enriquecimento das Informações de Contexto

Atualmente, consideramos como contexto, uma informação que indica sob que condição uma correspondência entre dois conceitos de ontologias distintas é estabelecida. Em suma, esta condição é representada pelo domínio (ou super classe do domínio) de uma propriedade ou ainda por restrições que utilizam operadores de comparação. Entretanto, a definição de contexto pode ser bem mais abrangente do que esta utilizada. Nesse sentido, podemos aprimorar o uso do contexto neste trabalho com o intuito de adicionar mais informações semânticas ao processo de reescritas, de forma que este passe a considerar aspectos específicos do ambiente, preferências e *feedback* do usuário. Além disso, podemos pensar em representar tais informações de contexto através de estrutuuras mais robustas, como por exemplo, através de grafos. Em [Reddy et. al 2010], por exemplo, os autores utilizam o que chamam de grafo de contexto para otimizar consultas submetidas sobre a *Web* de Dados Ligado (*Linked Data*).

- Extensão do processo para consideração de múltiplas fontes de dados (ontologias)

Neste aspecto, devemos considerar a reescrita de consultas para várias ontologias e a fusão dos resultados destas consultas. Mais especificamente, no que tange ao segundo aspecto, um problema bastante recorrente na *Web* Semântica e no cenário de Dados Interligados (*Linked Data*) consiste em determinar quando duas instâncias pertencentes a ontologias ou *datasets* distintos representam o mesmo objeto do mundo real. Este problema é chamado de *co-reference resolution*.

- Otimização das Consultas

Nesta linha, dois pontos principais merecem ser abordados com mais profundidade: (i) o desenvolvimento ou a adoção de um *benchmark* que possibilite a realização de testes mais massivos com as consultas, principalmente as que podem ser

reduzidas através da utilização das informações de contexto; (ii) uma análise mais profunda sobre otimização de consultas em SPARQL, de forma a possibilitar o desenvolvimento de uma estratégia mais refinada.

- Realização de melhorias na estratégia de reescrita

Uma vez que a estratégia proposta resolve o problema descrito, é possível realizar melhorias nesta estratégia, de forma a torná-la mais eficiente. Para isto, podemos verificar como se comportaria uma abordagem mista. Tal abordagem realizaria parte da expansão de conceitos na própria consulta e outra parte nos mapeamentos, conforme já é feito atualmente. A ideia é que conseguíssemos reduzir o conjunto dos mapeamentos, mas continuássemos sendo capazes de lidar com ontologias que possuem estruturas distintas. Por outro lado, podemos investigar com mais profundidade o problema de *Query Containment*, de modo a verificar se soluções neste sentido podem ser incorporadas.

Referências Bibliográficas

Adjiman, P., Chatalic, P., Goasdoué, F., Rousset, M., and Simon, L. (2006). Distributed reasoning in a peer-to-peer setting: Application to the semantic web. *Journal of Artificial Intelligence Research (JAIR)*, 25(1):269–314.

Adjiman, P., Goasdoué, F., and Rousset, M. (2007). Somerdfs in the semantic web. *Journal on Data Semantics VIII*, pages 158–181.

Akhtar, W., Kopecky, J., Krennwallner, T., and Polleres, A. (2008). Xsparql: Traveling between the xml and rdf worlds - and avoiding the xslt pilgrimage. In *Proceedings of the 5th European Semantic Web Conference (ESWC)*, Berlin, Heidelberg.

Alhajj, R. (2003). Extracting the extended entity-relationship model from a legacy relational database. *Information Systems*, 28(6):597–618.

An, Y., Borgida, A., and Mylopoulos, J. (2005). Constructing complex semantic mappings between xml data and ontologies. In *Proceedings of 4th International Semantic Web Conference (ISWC'05)*, pages 6–20, Galway, Ireland.

Arenas, M., Gutierrez, C., and P´erez, J. (2009). On the semantics of sparql. In Roberto De Virgilio, F. G. and Tanca, L., editors, *Semantic Web Information Management: A Model Based Perspective*, chapter 13. Spring, 1st edition.

Astrova, I. and Kalja, A. (2008). Automatic transformation of sql relational databases to owl ontologies. In *Proceedings of the Fourth International Conference on Web Information Systems and Technologies (WEBIST)*, pages 131–136, Madeira, Portugual.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. (2007). *The description Logic Handbook*. Cambridge University Press.

Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A. (2004). OWL web ontology language reference. W3C recommendation. Disponível em: http://www.w3.org/TR/owl-ref/. Acessado em 30 de junho de 2010.

Bernstein, A., Kiefer, C., and Stocker, M. (2007). Optarq: A sparql optimization approach based on triple pattern selectivity estimation. Technical report, University of Zurich.

Bikakis, N., Gioldasis, N., Tsinaraki, C., and Christodoulakis, S. (2009a). Querying xml data with sparql. In *Proceedings of the 20th International Conference on Database and Expert Systems Applications (DEXA)*, pages 372–381.

Bikakis, N., Gioldasis, N., Tsinaraki, C., and Christodoulakis, S. (2009b). The sparql2xquery framework. Technical Report. Disponível em: http://www.music.tuc.gr/reports/SPARQL2XQUERY.pdf. Acessado em 15 de junho de 2010.

Bilke, A. (2007). *Duplicate-based Schema Matching*. PhD thesis, Berlin University.

Bizer, C., Heath, T., and Berners-Lee, T. (2009). Linked data - the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22.

Boag, S., Chamberlin, D., Fernández, M. F., Florescu, D., Robie, J., and Siméon, J. (2007). Xquery 1.0: An xml query language. W3C recommendation. Disponível em: http://www.w3.org/TR/xquery/. Acessado em 10 de maio de 2010.

Bohring, H. and Auer, S. (2005). Mapping xml to owl ontologies. *Leipziger Informatik Tage*, 72:147–156.

Brickley, D. and Guha, R. (2004). Rdf vocabulary description language 1.0: Rdf schema. W3C recommendation. Disponível em: http://www.w3.org/TR/rdf-schema/. Acessado em 20 de junho de 2010.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., and Rosati, R. (2009). Ontologies and databases: The DL-lite approach. *Reasoning Web. Semantic Technologies for Information Systems*, pages 255–356.

Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. Y. (2000). What is query rewriting? In *Proceeding of the 7th International Workshop on Knowledge Representation meets Databases (KRDB)*, pages 17–27.

Calvanese, D., Giacomo, G. D., Lembo, D., Lenzerini, M., Poggi, A., and Rosati, R. (2008). Data integration through dl-litea ontologies. In *Semantics in Data and Knowledge Bases: Third International Workshop (SDKB 2008) - Revised Selected Papers*, pages 26–47, Nantes, France.

Calvanese, D., Giacomo, G. D., Lembo, D., Lenzerini, M., and Rosati, R. (2004). What to ask to a peer: Ontology-based query reformulation. In *Proceedings of the 9th*

International Conference on the Principles of Knowledge Representation and Reasoning (KR 2004), pages 469–478.

Cerbah, F. (2008). Learning highly structured semantic repositories from relational databases: the RDBtoOnto tool. In *Proceedings of the 5th European Semantic Web Conference on the Semantic Web (ESWC)*, pages 777–781. Springer-Verlag.

Chebotko, A., Lu, S., and Fotouhi, F. (2009). Semantics preserving sparql-to-sql translation. *Data & Knowledge Engineering*, 68(10):973–1000.

Clark, J. (1999). XSL transformations (XSLT). W3C recommendation. Disponível em: http://www.w3.org/TR/xslt. Acessado em: 30 de julho de 2010.

Connolly, D. (2007). Gleaning resource descriptions from dialects of languages. W3C recommendation. Disponível em: http://www.w3.org/TR/grddl/. Acessado em: 30 de julho de 2010.

Correndo, G., Salvadores, M., Millard, I., Glaser, H., and Shadbolt, N. (2010). Sparql query rewriting for implementing data integration over linked data. In *Proceedings of the International Workshop on Data Semantics (DataSem 2010) - In conjunction with the 13th International Conference on Extending Database Technology (EDBT)*, pages 1–11, Lousanne, Switzerland.

Cyganiak, R. (2005). A relational algebra for SPARQL. Technical report, HP-Labs Technical Report, HPL-2005-170. Disponível em: http://www.hpl.hp.com/techreports/2005/HPL-2005-170.html. Acessado em 05 de Agosto de 2010.

De Bruijn, J. and Polleres, A. (2004). Towards an ontology mapping specification language for the semantic web. Technical report, Technical Report DERI-2004-06-30, DERI. Disponível em: http://www.deri.at/publications/techpapers/documents/DERI-TR-2004-06-30.pdf. Acessado em: 05 de Agosto de 2010.

Droop, M., Flarer, M., Groppe, J., Groppe, S., Linnemann, V., Pinggera, J., Santner, F., Schier, M., Schopf, F., Staffler, H., and Zugal, S. (2008). Embedding XPATH queries into SPARQL queries. In *Proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS)*.

Elliott, B., Cheng, E., Thomas-Ogbuji, C., and Ozsoyoglu, Z. M. (2009). A complete translation from sparql into efficient sql. In *Proceedings of the International Database Engineering & Applications Symposium (IDEAS)*, pages 31–42.

Euzenat, J. and Shvaiko, P. (2007). *Ontology Matching*. Springer-Verlag New York Inc.

Farrell, J. and Lausen, H. (2007). Semantic annotations for wsdl and xml schema. W3C recommendation. Disponível em: http://www.w3.org/TR/sawsdl/. Acessado em 20 de julho de 2010.

Fernandes, D. Y. S. (2009). *Using Semantics to Enhance Query Reformulation in Dynamic Distributed Environments*. PhD thesis, Federal University of Pernambuco.

Garcia, R. and Celma, O. (2005). Semantic integration and retrieval of multimedia metadata. In *5th International Workshop on Knowledge Markup and Semantic Annotation (SemAnnot 2005) located at the 4rd International Semantic Web Conference (ISWC)*, pages 69–80, Galway, Ireland.

Ghawi, R. and Cullot, N. (2009). Building ontologies from XML data sources. In *Proceedings of the International Workshop on Modelling and Visualization of XML and Semantic Web Data (MoVIX) - In conjunction with the 20th International Conference on Database and Expert Systems Application (DEXA)*, pages 480 – 484, Linz, Austria.

Ghidini, C. and Serafini, L. (2006). Reconciling concepts and relations in heterogeneous ontologies. In *Proceedings of the 3rd European Semantic Web Conference (ESWC)*, pages 50–64, Budva, Montenegro.

Goasdoué, F. and Rousset, M. (2004). Answering queries using views: a KRDB perspective for the semantic web. *ACM Transactions on Internet Technology (TOIT)*, 4(3):255–288.

Green, T. J., Karvounarakis, G., Taylor, N. E., Biton, O., Ives, Z. G., and Tannen, V. (2007). Orchestra: facilitating collaborative data sharing. In *Proceedings of the ACM SIGMOD International Conference on Management Of Data*, pages 1131–1133.

Groppe, J., Groppe, S., and Kolbaum, J. (2009). Optimization of SPARQL by using CORESPARQL. In *Proceedings of the 11th International Conference on Enterprise Information Systems (ICEIS)*, pages 14–26, Milan, Italy.

Groppe, S., Groppe, J., Linnemann, V., Kukulenz, D., Hoeller, N., and Reinke, C. (2008). Embedding SPARQL into XQUERY/XSLT. In *Proceedings of the Symposium on Applied Computing (SAC)*, pages 2271–2278.

Hull, R. and Yoshikawa, M. (1990). ILOG: Declarative creation and manipulation of object identifiers. In *Proceedings of the 16th International Conference on Very Large Databases (VLDB)*, pages 455–468, Brisbane, Australia.

Ianni, G., Krennwallner, T., Martello, A., and Polleres, A. (2009). Dynamic querying of mass-storage RDF data with rule-based entailment regimes. In *International Semantic Web Conference*, pages 310–327.

Jing, Y., Jeong, D., and Baik, D.-K. (2009). SPARQL graph pattern rewriting for OWL-DL inference queries. *Knowledge and Information Systems*, 20(2):243–262.

Klien, E. (2008). *Semantic Annotation of Geographic Information*. PhD thesis, University of Muenster, Germany.

Klyne, G. and Carroll, J. J. (2004). Resource description framework (RDF): Concepts and abstract syntax. W3C recommendation. Disponível em: http://www.w3.org/TR/rdf-concepts. Acessado em 30 de junho de 2010.

Leme, L. A. P. P. (2009). *Conceptual Schema Matching based on Similarity Heuristics*. PhD thesis, PUC - Rio, Brazil.

Lenzerini, M. (2002). Data integration: A theoretical perspective. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 233–246.

Lloyd, J. (1987). *Foundations of Logic Programming*. Berlin: Springer-Verlag.

Lloyd, J. W. and Shepherdson, J. C. (1991). Partial evaluation in logic programming. *Journal of Logic Programming*, 11:217–242.

Lubyte, L. and Tessaris, S. (2009). Automatic extraction of ontologies wrapping relational data sources. In *Proceedings of the 20th International Conference on Database and Expert Systems Applications (DEXA)*, pages 128–142.

Lutz, M. (2006). *Ontology-based discovery and Composition of Geographic Information Services*. PhD thesis, Institut fur Geoinformatik, Germany.

Ma, L., Wang, C., Lu, J., Cao, F., Pan, Y., and Yu, Y. (2008). Effective and efficient semantic web data management over DB2. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1183–1194.

Madhavan, J. and Halevy, A. Y. (2003). Composing mappings among data sources. In *Proceedings of the 29th international conference on Very large databases (VLDB)*, pages 572–583.

Makris, K., Gioldasis, N., Bikakis, N., and Christodoulakis, S. (2010). Ontology mapping and sparql rewriting for querying federated RDF data sources. In *Proceedings of the 9th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE)*, Crete, Greece.

Manjunath, B., Salembier, P., and Sikora, T. (2002). *Introduction to MPEG-7: Multimedia Content Description Interface*. John Wiley & Sons Inc.

Mcbrien, P. and Poulovassilis, A. (2007). P2P query reformulation over both-as-view data transformation rules. In *Proceedings of the international conference on Databases, Information Systems and Peer-to-Peer computing (DBISP2P)*, pages 310-322.

Necib, C. B. (2007). *Ontology-based semantic query processing in database systems*. PhD thesis, Humboldt Universitat, Berlin.

Ng, W. S., Ooi, B. C., Kian-LeeTan, and AoyingZhou (2003). PeerDB: A p2p-based system for distributed data sharing. In *Proceedings of the 19th International Conference on Data Engineering (ICDE)*.

Nyulas, C., OConnor, M., and Tu, S. (2007). Datamaster - a plug-in for importing schemas and data from relational databases into protege. In *Proceedings of the 10th International Protege Conference*.

Pérez, J., Arenas, M., and Gutierrez, C. (2009). Semantics and complexity of sparql. *ACM Transactions on Database Systems (TODS)*, 34(3):1–45.

Pires, C. E. S. (2009). *Ontology-based Clustering in a Peer Data Management System*. PhD thesis, Federal University of Pernambuco.

Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., and Rosati, R. (2008). Linking data to ontologies. *Journal on data semantics X*, pages 133–173.

Polleres, A. (2007). From sparql to rules (and back). In *Proceedings of the 16th international conference on World Wide Web (WWW)*, pages 787–796.

Prud'hommeaux, E. and Seaborne, A. (2008). Sparql query language for rdf. w3c recommendation. Disponível em: http://www.w3.org/TR/rdf-sparql-query/. Acesso em 10 de setembro de 2010.

Quilitz, B. and Leser, U. (2008). Querying distributed rdf data sources with SPARQL. In *Proceedings of the 5th European semantic web conference on the semantic web (ESWC)*, pages 524–538.

Reddy, B.R.K. and Kumar, P.S. (2010). Optimizing SPARQL queries over the Web of Linked Data. In *Proceeding of the International Workshop on Semantic Data Management (SemData) - In conjunction with the 36th International Conference on Very Large Data Bases (VLDB)*.

Sacramento, E. R., Vidal, V. M., Macêdo, J. A., Lóscio, B. F., Lopes, F. L. R., and Casanova, M. A. (2010a). Towards automatic generation of application ontologies. *Journal of Information and Data Management (JIDM)*, 1(3):535–551.

Sacramento, E. R., Vidal, V. M., Macêdo, J. A., Lóscio, B. F., Lopes, F. L. R., Casanova, M. A., and Lemos, F. (2010b). Towards automatic generation of application ontologies. In *Proceeding of the 12th International Conference on Enterprise Information Systems - ICEIS*, Funchal, Madeira - Portugal.

Schenk, S. (2007). A SPARQL semantics based on datalog. In *Proceedings of the 30th annual German conference on Advances in Artificial Intelligence*, pages 160–174.

Schmidt, M., Meier, M., and Lausen, G. (2010). Foundations of SPARQL query optimization. In *Proceedings of the 13th International Conference on Database Theory (ICDT)*, pages 4–33.

Sonia, K. and Khan, S. (2008). R2O transformation system: relation to ontology transformation for scalable data integration. In *Proceedings of the 2008 International Symposium on Database Engineering & Applications (IDEAS)*, pages 291–295.

Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C., and Reynolds, D. (2008). SPARQL basic graph pattern optimization using selectivity estimation. In *Proceeding of the 17th international conference on World Wide Web (WWW)*, pages 595–604.

Stuckenschmidt, H., Giunchiglia, F., and van Harmelen, F. (2005). Query processing in ontology-based peer-to-peer systems. In Tamma, V., Cranefield, S., Finin, T., and Willmott, S., editors, *Ontologies for Agents: Theory and Experiences*. Birkhauser.

Trinh, Q., Barker, K., and Alhajj, R. (2006). RDB2ONT: A tool for generating OWL ontologies from relational database systems. In *Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW)*, pages 170–178, Washington, DC, USA. IEEE Computer Society.

Tsinaraki, C. and Bikakis, N. (2007). Interoperability of XML schema applications with OWL domain knowledge and semantic web tools. In *Proceedings of the International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE)*, pages 850–869.

Vidal, V. M., Sacramento, E. R., Macêdo, J. A., and Casanova, M. A. (2009). An ontology-based framework for geographic data integration. In *Proceedings of the 3rd International Workshop on Semantic and Conceptual Issues in Geographic Information Systems (SeCoGIS) - In conjunction with the 28th International Conference on Conceptual Modeling (ER)*, pages 337–346, Gramado, Brazil.

Wache, H., V¨ogele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., and Hubner, S. (2001). Ontology-based integration of information - a survey of existing approaches. In *Proceedings of the International Workshop on Ontologies and Information Sharing - In conjunction with the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 108–117, Seattle, USA.

Wiederhold, G. (1992). Mediators in the architecture of future information systems. *Computer*, 25(3):38–49.

Xiao, H. and Cruz, I. F. (2006). Ontology-based query rewriting in peer-to-peer networks. In *Proceedings of the 2nd International Conference on Knowledge Engineering and Decision Support*.

Yu, C. and Popa, L. (2004). Constraint-based XML query rewriting for data integration. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 371–382.

Apêndice A
Conceitos sobre RDF e SPARQL

Conceitos Fundamentais de RDF/OWL

O RDF (*Resource Description Framework*) é um modelo de dados para representação de informações sobre recursos na *Web*. De acordo com [Klyne and Carroll 2004], um recurso é tudo que possui uma identidade, qualquer objeto sobre o qual se deseje "falar". Tal objeto pode ser uma entidade digital (uma página *Web*, um documento eletrônico ou um serviço), uma entidade física (como um livro) ou ainda uma coleção de outros recursos. Todo recurso possui uma *URI* (*Uniform Resource Identifier*) que o identifica unicamente. O bloco fundamental para qualquer construção em RDF é definido por uma tripla, chamada de *sentença*, conforme descrito a seguir.

Definição 1.1. Sentença (Tripla) RDF **[**Arenas et al. 2009**]**. Seja *I* um conjunto de URIs, *L* um conjunto de *literais* RDF e *B* um conjunto de *Blank Nodes*. Um literal RDF é um tipo primitivo de dados expresso através dos *datatypes* do XML *Schema,* tais como inteiros ou *strings*. Já os *Blank Nodes* são nós anônimos que identificam localmente um determinado objeto na ontologia. Os conjuntos *I*, *U* e *B* são disjuntos dois a dois. Uma *sentença* RDF é uma tripla $(S, P, O) \in (I \cup B) \times I \times (I \cup B \cup L)$, onde:

- *S* é o *sujeito* da sentença.
- *P* é o *predicado* da sentença, o qual é uma propriedade que denota um relacionamento binário entre o sujeito e o objeto.
- *O* é o *objeto* da sentença, que corresponde ao valor da propriedade *P* no relacionamento.

Note que uma sentença pode ser representada através de um grafo dirigido, onde o sujeito e o objeto são os nós do grafo, enquanto as propriedades (relacionamentos) são as arestas. Um *grafo RDF* é caracterizado por um conjunto de sentenças.

Embora RDF seja uma linguagem extremamente flexível, uma de suas limitações consiste na ausência de construtores que permitam a definição da semântica de um domínio específico ou de uma aplicação. Isto significa que podemos representar os dados em RDF, porém não há meios para se definir um vocabulário que descreva tais dados. Por este motivo, foi criada uma extensão de RDF, chamada de RDF *Schema* ou RDFS [Brickley and Guha 2004], que fornece primitivas básicas para a modelagem de classes (*rdfs:Class*), subclasses (*rdfs:subClassOf*), propriedades (*rdfs:Property*), subpropriedades (*rdfs:subPropertyOf*) e instâncias. Em conjunto com as definições de

propriedades, são utilizados os construtores *rdfs:domain* e *rdfs:range*. O domínio de uma propriedade (*rdfs:domain*) limita a classe (ou classes) sobre os quais a propriedade pode ser aplicada. Já o *range* é utilizado para indicar quais valores podem ser atribuídos a uma determinada propriedade (podem ser valores literais ou outras classes).No que se refere às instâncias, estas podem ser definida conforme descrito a seguir.

Definição 1.2. Instâncias **[**Leme 2009**].** Uma *instância* de uma classe *C* é um recurso *R* para o qual há uma sentença do tipo *(R, rdf:type, C*). Tal sentença pode estar presente de maneira explícita na ontologia, ou ainda, ser inferida a partir de outras informações existentes. Um recurso pode ser instância de uma ou mais classes. Para definir que uma instância *R* de uma classe possui uma propriedade *P* com o valor *V*, é necessário definir uma sentença da forma (*R, P, V*).

A linguagem OWL [Bechhofer et al. 2004], a qual é uma recomendação do W3C (*World Wide Web Consortium*) para a representação de ontologias, pode ser vista como uma extensão de RDF/RDFs, agregando a esta linguagem a expressividade dos construtores lógicos. Com isso, temos que uma descrição em OWL é ainda uma descrição em RDF, pois todo o vocabulário OWL é especificado em RDF. Dentre os construtores adicionais presentes nesta linguagem, temos os que definem componentes básicos, tais como as classes (*owl:Class*) e as propriedades. No caso das propriedades, estas são divididas em duas categorias: propriedades de objeto (owl: *ObjectProperty)*, que descrevem um relacionamento entre duas classes, e propriedades de tipo de dado (*owl:DatatypeProperty*), que relacionam um indivíduo a um valor literal.

Além disso, OWL fornece construtores para formalização de relacionamentos mais sofisticados (equivalência, interseção e união de classes, por exemplo), bem como para definição de cardinalidade e outras construções para esquemas complexos. A linguagem OWL divide-se ainda em três sublinguagens, que diferem entre si quanto à expressividade e à complexidade: *OWL Lite, OWL DL*e *OWL Full*. A primeira consiste na versão mais simples e menos expressiva, possuindo, dessa forma, a menor complexidade computacional. Já OWL DL busca aliar expressividade à completude computacional e à decidibilidade, impondo, para isto, algumas restrições sobre o uso de certos construtores. Este dialeto é assim chamado devido à sua correspondência com a Lógica Descritiva. Por fim, *OWL Full* provê a máxima expressividade da linguagem OWL, permitindo uma vasta gama de representações semânticas, porém sem garantias computacionais para todas as características. Uma ontologia definida em uma espécie de

OWL menos expressiva está contida dentro das mais expressivas. A recíproca, no entanto, não é verdadeira.

A Linguagem SPARQL

SPARQL [Prud'hommeaux and Seaborne 2008] (acrônimo de *SPARQL Protocol and RDF Query Language*) é uma linguagem de consulta projetada para manipular dados na forma de triplas. Esta linguagem consiste na atual recomendação do W3C para consultas sobre ontologias e grafos RDF, os quais podem estar distribuídos em diversas fontes, armazenados nativamente ou ainda disponíveis através de visões.

Com isso, SPARQL tem como característica básica a facilidade para realização de *matching* entre padrões de grafos, bem como para a extração de subgrafos de uma ontologia. A estrutura geral da linguagem é composta por três blocos fundamentais (ilustrados na Figura 2.1):

- **Formato do Retorno:** especifica o formato final no qual os resultados devem ser retornados para o usuário. SPARQL dispõe de quatro maneiras para exibição dos dados: (i) na forma de tabela, assim como SQL, através da cláusula SELECT; (ii) como um grafo construído através de *templates*, onde as variáveis são substituídas por seus respectivos valores, utilizando a cláusula CONSTRUCT; (iii) uma resposta booleana (*true/false*), por meio da cláusula ASK; (iv) um grafo RDF descrevendo os recursos encontrados (DESCRIBE).
- **Cláusula do *Dataset*:** especifica quais fontes RDF (*datasets*) devem ser consultadas.
- **Cláusula *Where*:** contém o padrão de grafo a ser avaliado com respeito ao *dataset*. Este padrão pode conter expressões mais complexas, incluindo operadores de união, aninhamentos, filtros, entre outros.

Opcionalmente, é possível aplicar *modificadores de solução*, tais como DISTINCT, ORDER BY e LIMIT, sobre os resultados obtidos. Como exemplo, considere a consulta apresentada na Figura 2.2, que recupera "*o nome e o email das pessoas que vivem no Brasil"*.

A primeira linha corresponde ao prólogo da consulta, que especifica o *namespace*, com sua respectiva abreviação, a ser utilizado no corpo desta consulta. Neste caso, este *namespace* refere-se à ontologia FOAF[16]. A palavra chave SELECT indica que a consulta retornará uma tabela com duas colunas, correspondentes aos

[16]http://www.foaf-project.org/

valores obtidos através do casamento das variáveis ?*nome* e *?email* contra o grafo apontado na cláusula FROM (*minhaFonteDados.rdf*). Este casamento ocorre de acordo com o padrão descrito na cláusula WHERE. Note que uma *string* iniciando com o símbolo "?" denota uma variável em SPARQL (o símbolo "$" também identifica uma variável).

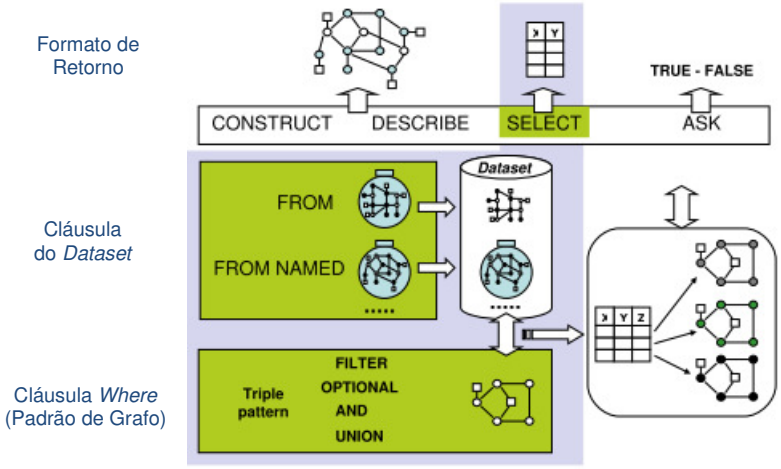


Figura 0.1. Forma geral de uma consulta SPARQL (adaptada de [Arenas et al. 2009])

```
PREFIX foaf: http://xmlns.com/foaf/0.1/

SELECT DISTINCT?nome  ?email
FROM <minhaFonteDados.rdf>
WHERE {
      ?x foaf:name ?nome .
      ?x foaf:mbox ?email .
      ?x foaf:country ?pais .
FILTER regex(?pais, "Brazil")
}
ORDER BY?nome
```

Figura 0.2. Exemplo de uma consulta em SPARQL

Além disso, em SPARQL, as variáveis possuem um escopo global, ou seja, uma vez que aparecem, são válidas em toda a extensão da consulta. Ainda na Figura 1.3, a cláusula WHERE é composta por um padrão com três triplas: ?*x foaf:name ?nome*, *?x foaf:mbox ?email* e *?x foaf:country ?pais*. As palavras-chave DISTINCT, FILTER e ORDER BY possuem a seguintes finalidades: DISTINCT indica que somente valores distintos devem ser retornados; FILTER restringe a soluçãoaos valores que obedecem à expressão contida no filtro; ORDER BY especifica como classificar a lista de resultados.

A seguir, serão explanadas as principais características, componentes e a semântica da referida linguagem. Maiores detalhes podem ser encontrados em

[Prud'hommeaux and Seaborne 2008] e [Arenas et al. 2009].

Especificação dos *Datasets*

Em SPARQL, um *dataset DS* define o escopo de avaliação da consulta, podendo este envolver informações oriundas de um ou mais ontologias. Dessa forma, $DS = (G, (g_1, G_1),..., (g_k, G_k))$, onde:

- *G* é um *grafo default* consistindo do *merge* dos grafos especificados por suas IRIs[17] na cláusula FROM;
- O conjunto de pares (*IRI, grafo*) identifica os *grafos nomeados*. Para cada um destes grafos há uma cláusula FROM NAMED associada.

Os grafos nomeados tem como função permitir que o grafo alvo da consulta seja modificado localmente, na cláusula WHERE, através da utilização da palavra-chave GRAPH. Tal palavra-chave deve referir-se a estes grafos nomeados e, caso não seja utilizada, o grafo alvo da consulta consiste somente do grafo *default*.

Padrões de Grafos (GP)

Baseado na descrição de grafos RDF apresentada anteriormente, SPARQL utiliza o casamento (*matching*) de padrões de grafos para expressar suas consultas. Estes padrões de grafos são definidos recursivamente, sendo possível construir padrões complexos a partir dos padrões mais simples, em conjunto com os operadores da álgebra SPARQL (*AND, UNION, OPT, GRAPH e FILTER*).

Definição 1.3. Padrão de Grafo (GP). Um Padrão de Grafo SPARQL é definido recursivamente da seguinte forma [Arenas et al. 2009]:

(1) Um Padrão de Tripla (TP) é um padrão de grafo.

(2) Se P1 e P2 são padrões de grafo, então as expressões (P1 AND P2), (P1 OPT P2) e (P1 UNION P2) são padrões de grafo.

(3) Se P é um padrão de grafo e *i* é uma IRI ou uma variável, então (GRAPH *i* P) é um padrão de grafo.

(4) Se P é um padrão de grafo e R é uma expressão de filtro SPARQL, então (P FILTER R) é um padrão de grafo.

Padrões de Triplas (TP)

[17]Uma IRI [RFC3987] é uma versão internacionalizada de uma URI. Neste trabalho, URI e IRI serão utilizadas com o mesmo significado.

Correspondem aos padrões simples na forma de triplas RDF, conforme apresentado na Definição 1.1. A única diferença entre uma *tripla* e um *padrão de tripla*consiste no fato de um padrão de tripla admitir a existência de variáveis em seus componentes. Ou seja, $(S, P, O) \in (I \cup B \cup V) \times (I \cup V) \times (I \cup B \cup L \cup V)$.

Padrões de Grafos Básicos (BGP)

É uma sequência de padrões de triplas, possivelmente com expressões de FILTRO, combinados através do operador de conjunção (AND, representado sintaticamente por "."). Isto significa que o filtro não gera um novo BGP.

Grafos Opcionais (OPT)

Padrões de grafos básicos permitem a construção de consultas nas quais todo o padrão deve estar presente no *dataset* para que haja uma solução. Visando permitir que uma solução seja aceita, emboranem todo o padrão de grafo seja recuperado, o operador de OPT foi desenvolvido. Através de sua utilização, se o trecho opcional não estiver presente no *dataset*, a informação referente a tal trecho não é retornada, mas isto não elimina o restante da solução. Sintaticamente, o operador OPT é expresso através da palavra-chave OPTIONAL, da seguinte forma: *{OPTIONAL { GP }}*, a qual é, algebricamente, equivalente a *(( ) OPTIONAL (GP))*.

A combinação do operador OPT com restrições de FILTRO permite a construção de consultas mais sofisticadas, tais como as que expressam negação, através de *negação por falha* [Lloyd 1987; Prud'hommeaux and Seaborne 2008].

Grafos Alternativos (UNION)

Permitem combinar padrões de grafos de forma que um dos padrões alternativos possa se recuperado na fonte de dados. Se mais de uma das alternativas for obtida, então todas as soluções possíveis são retornadas. Padrões alternativos são especificados através do operador de união (UNION), da seguinte forma: *{{ GP } UNION { GP }}*.

Grafos Nomeados (GRAPH)

A definição de grafos nomeados foi apresentada na Seção 1.2.1. Uma vez que a consulta possua tais grafos, o operador GRAPH é utilizado para especificar sobre quais dos

datasets o padrão entre chaves deve ser buscado. Este operador é expresso da seguinte maneira: *GRAPH <Var ou IRI> {GP}*.

Filtros

Em SPARQL, os filtros (FILTER) contêm restrições que limitam o resultado da consulta para os valores nos quais a expressão contida no filtro é avaliada como verdadeira. Tais restrições são válidas para todo o grupo limitado por { }, no qual o filtro aparece. SPARQL permite que combinações booleanas possam ser utilizadas dentro das expressões de filtro, bem como fornece um conjunto de operadores (unários, binários e ternários) que auxiliam na construção das restrições. A lista completa de combinações e operadores pode ser encontrada em [Prud'hommeaux and Seaborne 2008].

Modificadores de Solução

Permitem gerenciar o resultado da consulta, criando uma nova sequência que obedece às restrições do modificador aplicado. Em SPARQL, os modificadores existentes são os listados a seguir. Alguns destes desempenham papel semelhante em SQL.

- ORDER BY: ordena a solução de maneira crescente (ASC) ou decrescente (DESC).
- PROJECTION: reduz a solução para o subconjunto de variáveis presentes na cláusula SELECT.
- DISTINCT: elimina duplicatas da solução.
- REDUCED: permite que qualquer solução não-única seja eliminada. Ao contrário do DISTINCT, não necessariamente elimina soluções repetidas.
- OFFSET: indica que a solução gerada deve ser exibida a partir de um número de soluções especificado.
- LIMIT: restringe o número de soluções apresentadas.

Apêndice B

Algoritmo de Unificação

Entrada: Duas expressões t_1 e t_2
Saída: O *mgu* destas expressões, se elas forem unificáveis.
Caso contrário, retorna *Falha*.

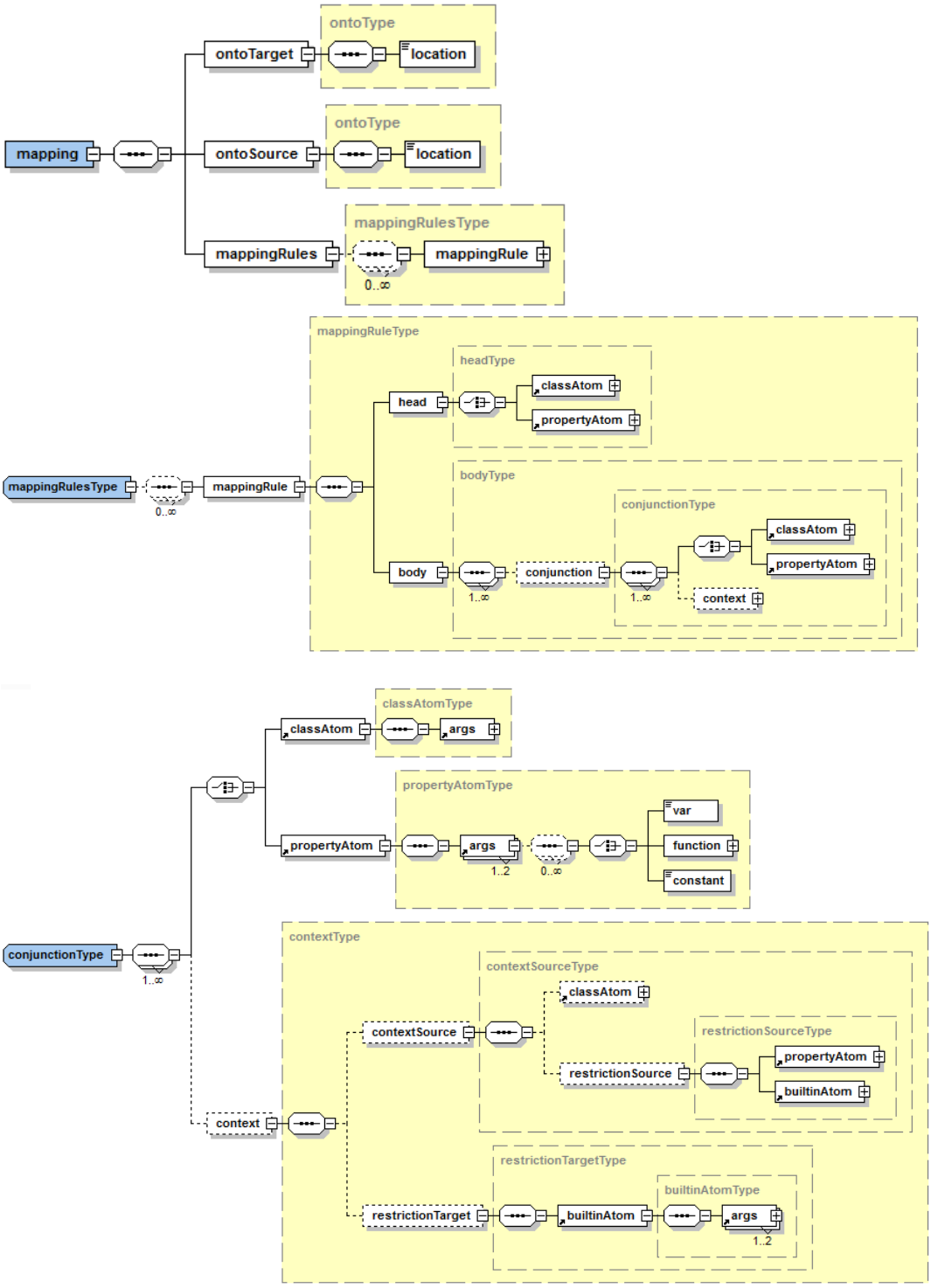
Unifica(t_1,t_2)

```
1.  Se t1 ou t2 é uma constante ou uma variável Então
2.    Se t1 = t2 Então
3.      Retorne { }
4.    Senão
5.      Se t2 for uma variável Então
6.        Retorne {t2/t1}
7.      Senão
8.        Se t1 for uma variável Então
9.          Retorne {t1/t2}
10.       Senão
11.         Retorne Falha
12. Senão
13.   Seja t1 = φ1(t11, ..., t1n) e t2 = φ2(t11, ..., t2n')
14.   Se  φ1 = φ2 e n = n' Então
15.     Θ ← ∅
16.     Para i = 1 até n Faça
17.       σ ← Unifica(t1i, t2i)
18.       Se θ = Falha Então
19.         Retorne Falha
20.       Senão
21.         Θ ← Θσ
22.     Retorne Θ
23.   Senão
24.     Retorne Falha
```

Fim *Unifica*

Apêndice C

Modelo XML dos Mapeamentos

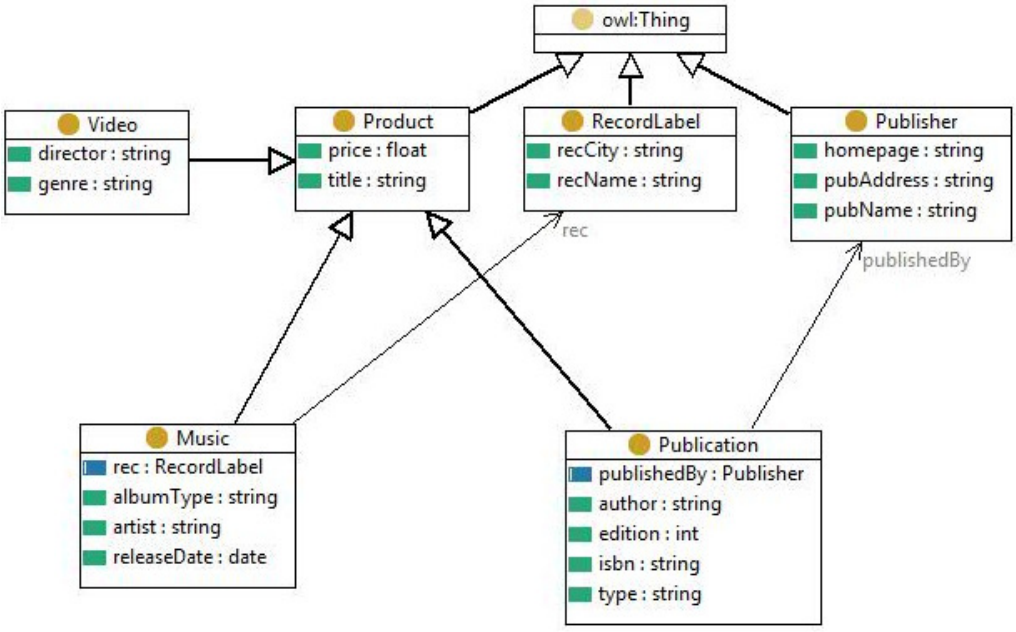


Apêndice D

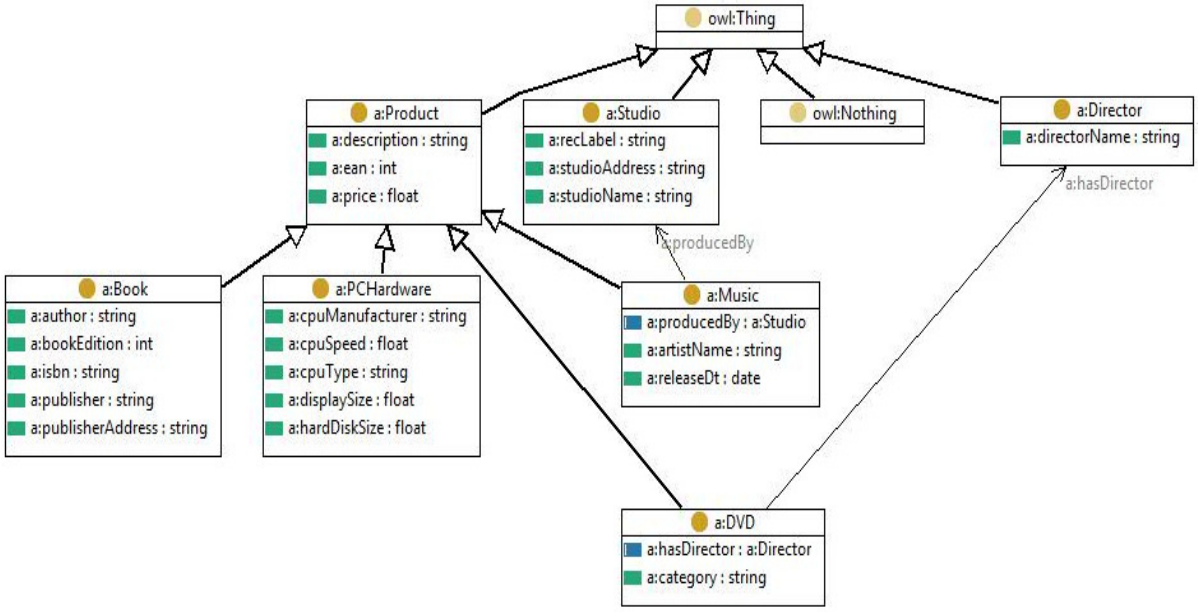
Trechos Relevantes das Ontologias Utilizadas nos Testes

Ontologias do Domínio de Vendas

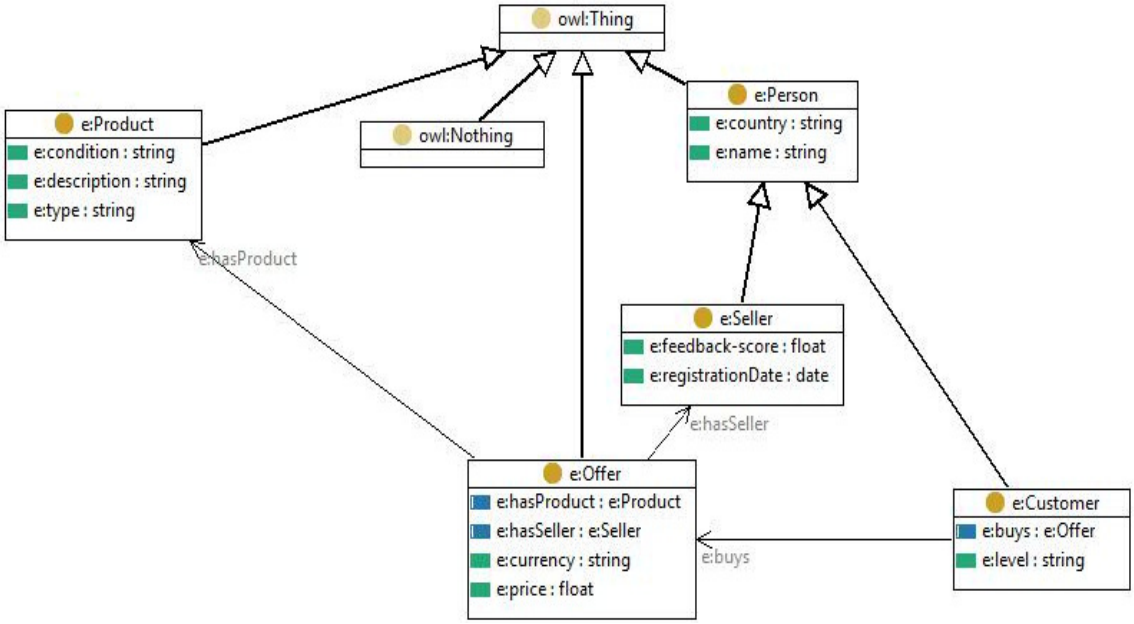
- *Sales.owl*



- *Amazon.owl*

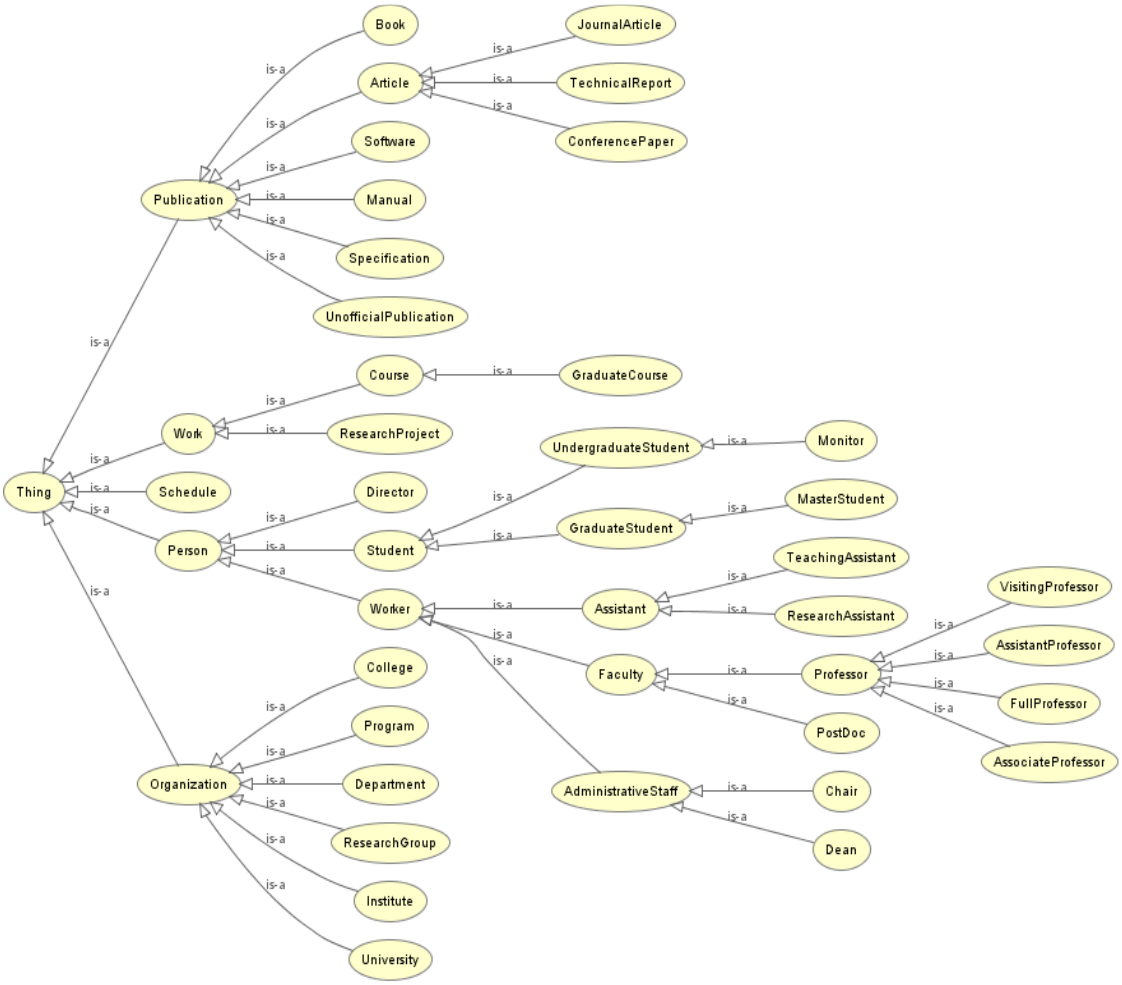


- *Ebay.owl*


owl:Thing
e:Product
e:condition : string
e:description : string
e:type : string
owl:Nothing
e:Person
e:country : string
e:name : string
e:hasProduct
e:Seller
e:feedback-score : float
e:registrationDate : date
e:hasSeller
e:Offer
e:hasProduct : e:Product
e:hasSeller : e:Seller
e:currency : string
e:price : float
e:buys
e:Customer
e:buys : e:Offer
e:level : string

Ontologias do Domínio de Educação

- *UnivBench.owl – Classes*[18]



[18] A Ontologia UnivBench.owl completa encontra-se disponível para download em: www.lia.ufc.br/~fernanda.ligia/SQUOL

- *Semiport.owl – Classes*[19]

[19] A Ontologia Semiport.owl completa encontra-se disponível para download em: www.lia.ufc.br/~fernanda.ligia/SQUOL

Apêndice E

Mapeamentos entre as as Ontologias

Domínio de Vendas

Ontologia Alvo: *Sales.owl* – **Ontologia Fonte:** *Amazon.owl*
Classes da Ontologia Alvo
s:Video(x) ← a:DVD(x) s:Music(x) ← a:Music(x) s:Publication(x) ← a:Book(x) s:Product(x) ← a:Book(x); a:DVD(x); a:Music(x) s:Publisher(*fPublisher(y)*) ← a:publisher(x,y), a:Book(x) s:RecordLabel(*fRLabel(y)*) ← a:producedBy(x,z), a:recLabel(z,y), a:Music(x)
Propriedades da Ontologia Alvo

s:type(x,k) ← a:Book(x), k = “book”;
s:artist(x,y) ← a:artistName(x,y), a:Music(x)
s:releaseDate(x,y) ← a:releaseDt(x,y), a:Music(x)
s:author(x,y) ← a:author(x,y), a:Book(x)
s:edition(x,y) ← a:bookEdition(x,y), a:Book(x)
s:isbn(x,y) ← a:isbn(x,y), a:Book(x)
s:genre(x,y) ← a:category(x,y), a:DVD(x)
s:price(x,y) ← a:price(x,y), a:Book(x);
a:price(x,y), a:DVD(x);
a:price(x,y), a:Music
s:title(x,y) ← a:description(x,y), a:Book(x);
a:description(x,y), a:DVD(x);
a:description(x,y), a:Music(x);
s:director(x,y) ← a:hasDirector(x,z), a:directorName(z,y), a:DVD(x);
s:pubName(*fPublisher(y)*,y) ← a:publisher(x,y), a:Book(x)
s:pubAddress(*fPublisher(y)*,z) ← a:publisherAddress(x,z), a:publisher(x,y), a:Book(x)
s:publishedBy(x,*fPublisher(y)*) ← a:publisher(x,y), a:Book(x)
s:recName(*fRLabel(y)*,y) ← a:recLabel(z,y), a:Music(x)
s:rec(x,*fRLabel(y)*) ← a:producedBy(x,z), a:recLabel(z,y), a:Music(x)

Ontologia Alvo: *Sales.owl* – **Ontologia Fonte:** *Ebay.owl*
Classes da Ontologia Alvo
s:Music(x) ←e:Product(x), e:type(x,k), k= “music” s:Publication(x)←e:Product(x), e:type(x,k), k= “book” s:Product(x) ← e:Product(x), e:type(x,k), k= “book”; e:Product(x), e:type(x,k), k= “music”
Propriedades da Ontologia Alvo
s:type(x,y)← e:Product(x), e:type (x,k), k= “book”, y= “book” s:title(x,y) ← e:description(x,y),s:Product(x), e:Product(x), e:type(x,k), k= “book”; e:description(x,y); e:Product(x), e:type(x,k), k= “music”

Ontologia Alvo: *Sales.owl* – **Ontologia Fonte:** *Ebay.owl*
Classes da Ontologia Alvo
e:Product(x) ← a:Book(x); a:Music(x) e:type(x,y)←a:Book(x), y= “book”; a:Music(x), y=”music”
Propriedades da Ontologia Alvo
e:description(x,y) ←a:description(x,y), a:Book(x); a:description(x,y), a:Music(x)

Domínio de Educação

Ontologia Alvo: *UnivBench.owl* – **Ontologia Fonte:** *Semiport.owl*

Classes da Ontologia Alvo

u:Organization(x) ← s:Organization(x); s:Department(x); s:ResearchGroup(x); s:University(x)
u:University(x) ← s:University(x)
u:Department(x) ← s:Department(x)
u:ResearchGroup(x) ← s:ResearchGroup(x)
u:TechnicalReport(x) ← s:TechnicalReport(x)
u:Book(x) ← s:Book(x)
u:UnofficialPublication(x) ← s:UnofficialPublication(x)
u:Manual(x) ← s:Manual(x)
u:Article(x) ← s:Article(x); s:TechnicalReport(x)
u:Publication(x) ← s:Article(x); s:Book(x); s:Manual(x); s:TechnicalReport(x);
s:UnofficialPublication(x)
u:AssistantProfessor(x) ← s:AssistantProfessor(x)
u:FullProfessor(x) ← s:FullProfessor(x)
u:Professor(x) ← s:AssistantProfessor(x); s:FullProfessor(x)
u:Faculty(x) ← s:AssistantProfessor(x); s:FullProfessor(x)
u:AdministrativeStaff(x) ← s:AdministrativeStaff(x)
u:Worker(x) ← s:AssistantProfessor(x); s:FullProfessor(x); s:AdministrativeStaff(x)
u:GraduateStudent(x) ← s:GraduateStudent(x)
u:UndergraduateStudent(x) ← s:UndergraduateStudent(x)
u:Student(x) ← s:Student(x); s:GraduateStudent(x); s:UndergraduateStudent(x)
u:Person(x) ← s:AssistantProfessor(x); s:FullProfessor(x); s:AdministrativeStaff(x); s:Student(x);
s:GraduateStudent(x); s:UndergraduateStudent(x)
u:ResearchProject(x) ← s:ResearchProject(x)
u:Work(x) ← s:ResearchProject(x)
u:Course(*fCourse*(y)) ← course(x,y) s:UndergraduateStudent(x)

Propriedades da Ontologia Alvo

Propriedades de Tipo de Dado:

u:emailAddress(x,y) ← s:emailAddress(x,y), u:Person(x) *
u:personName(x,y) ← s:name(x,y), u:Person(x) *
u:telephone(x,y) ← s:telephone(x,y), u:Person(x) *
u:pubTitle(x,y) ← s:pubTitle(x,y), u:Publication(x)*

u:courseName(*fCourse(y)*,y) ← s:course(x,y), s:Course(x)

Propriedades de Objeto:

u:takesCourse(x, *fCourse(y)*) ← s:course(x,y) s:UndergraduateStudent(x)

u:affiliateOf(x,y) ← s:memberOfOrganization(x,y), u:Person(x)*

u:affiliatedOrganizationOf(x,y) ← s:isOf(x,y), s:University(x)

u:member(x,y) ← s:isCoordinated(x,y), s:ResearchGroup(x);
s:member(x,y), s:Organization(x);
s:student(x,y), s:University(x)

u:memberOfOrganization(x,y) ← s:memberOfOrganization(x,y), u:Person(x);*
s:headOfGroup(x,y), u:Worker(x);*
s:coordinates(x,y), u:Faculty;*
s:studiesAt(x,y), s:Student(x)

u:orgPublication(x,y) ← s:publishes(x,y), s:Organization

u:publicationAuthor(x,y) ← s:publicationAuthor(x,y), u:Publication(x)*

u:subOrganizationOf(x,y) ← s:isOf(x,y), s:Departament(x)

u:worksForOrganization(x,y) ← s:coordinates(x,y), u:Faculty;*
s:headOfGroup(x,y), u:Worker(x)*

u:advises(x,y) ← s:advises, s:FullProfessor(x)

Obs:* Os predicados, do corpo das regras, que possuírem o *namespace* "u:" são apenas abreviações indicando que no local deles deve ser colocado o corpo da regra para qual eles apontam. Por exemplo, onde há *u:Faculty* do lado direito, considere *s:AssistantProfessor(x); s:FullProfessor(x)

Ontologia Alvo: *Semiport.owl* – **Ontologia Fonte:** *UnivBench.owl*

Classes da Ontologia Alvo

s:Department(x) ← u:Department(x)

s:ResearchGroup(x) ← u:ResearchGroup(x)

s:University(x) ← u:University(x)

s:Organization(x) ← u:Department(x); u:ResearchGroup(x); u:University(x)

s:TechnicalReport(x) ← u:TechnicalReport(x)

s:Book(x) ← u:Book(x)

s:UnofficialPublication(x) ← u:UnofficialPublication(x)

s:Manual(x) ← u:Manual(x)

s:Article(x) ← u:Article(x)

s:Publication(x) ← u:Article(x); u:Book(x); u:Manual(x); u:TechnicalReport(x);
u:UnofficialPublication(x)

s:FullProfessor(x) ← u:FullProfessor(x)

s:AssistantProfessor(x) ← u:AssistantProfessor(x)

s:Faculty(x) ← u:AssistantProfessor(x); u:FullProfessor(x)

s:AdministrativeStaff(x) ← u:AdministrativeStaff(x)

s:Worker(x) ← u:AssistantProfessor(x); u:FullProfessor(x); u:AdministrativeStaff(x)

s:UndergraduateStudent(x) ← u:UndergraduateStudent(x)

s:GraduateStudent(x) ← u:GraduateStudent(x)

s:Student(x) ← u:Student(x); u:GraduateStudent(x); u:UndergraduateStudent(x)

s:Person(x) ← u:AssistantProfessor(x); u:FullProfessor(x); u:AdministrativeStaff(x);
u:Student(x); u:GraduateStudent(x); u:UndergraduateStudent(x)

s:ResearchProject(x) ← u:ResearchProject(x)

s:Project(x) ← u:ResearchProject(x)

Propriedades da Ontologia Alvo

Propriedades de Tipo de Dado:

s:emailAddress(x,y) ← u:emailAddress(x,y), s:Person(x) *

s:name(x,y) ← u:personName(x,y), s:Person(x) *

s:telephone(x,y) ← u:telephone(x,y), s:Person(x) *

s:pubTitle(x,y) ← u:pubTitle(x,y), s:Publication(x)*

s:course(x,y) ← u:takesCourse(x,z), u:courseName(z,y) u:UndergraduateStudent(x)

Propriedades de Objeto:

s:headOfGroup(x,y) ← u:headOfOrganization(x,z), s:Worker(x)*

s:memberOfOrganization(x,y) ← u:memberOfOrganization(x,y), s:Person(x); *
u:worksForOrganization(x,y), s:Person(x)*

s:member(x,y) ← u:member(x,y), s:Organization(x)*

s:publicationAuthor(x,y) ← u:publicationAuthor(x,y), s:Publication(x)*

s:publishes(x,y) ← u:orgPublication(x,y), s:Organization(x)*

s:carriesOut(x,y) ← u:researchProject(x,y), u:ResearchGroup(x)

s:advises(x,y) ← u:advises, u:FullProfessor(x)

***Obs:* Os predicados, do corpo das regras, que possuírem o *namespace* "s:" são apenas abreviações indicando que no local deles deve ser colocado o corpo da regra para qual eles apontam. Por exemplo, onde há *s:Worker* do lado direito, considere u:AssistantProfessor(x); u:FullProfessor(x); u:AdministrativeStaff(x)**

Apêndice F

Consultas Testadas – Ontologias do Domínio de Vendas

Q_1 (Alvo: *Sales* – Fonte: *Amazon*)

Linguagem Natural

Retorne todos os Produtos.

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://www.owl-ontologies.com/sales.owl#>

SELECT ?prod
FROM <Sales.owl>
WHERE
{
   ?prod rdf:type s:Product
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>

SELECT ?prod
FROM <Amazon.owl>
WHERE
{
   {?prod rdf:type a:DVD}
   UNION
   {?prod rdf:type a:Music}
   UNION
   {?prod rdf:type a:Book}
}
```

Consulta reescrita oriunda de um processo básico (Q_{BAS})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>

SELECT ?prod
FROM <Amazon.owl>
WHERE
{
   {?prod rdf:type a:DVD}
   UNION
   {?prod rdf:type a:Music}
   UNION
   {?prod rdf:type a:Book}
}
```

Q_2 (Alvo: *Sales* – Fonte: *Amazon*)

Linguagem Natural

Retorne todas as Publicações

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://www.owl-ontologies.com/sales.owl#>

SELECT ?pub
FROM <Sales.owl>
WHERE
{
   ?pub rdf:type s:Publication
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>

SELECT ?pub
FROM <Amazon.owl>
WHERE
{
   ?pub rdf:type a:Book
}
```

Consulta reescrita oriunda de um processo básico (Q_{BAS})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>

SELECT ?pub
FROM <Amazon.owl>
WHERE
{
   ?pub rdf:type a:Book
}
```

Q_3 (Alvo: *Sales* – Fonte: *Amazon*)

Linguagem Natural

Retorne todos os Livros

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://www.owl-ontologies.com/sales.owl#>

SELECT ?book
FROM <Sales.owl>
WHERE
{
   ?book rdf:type s:Publication .
   ?book s:type ?y
   FILTER(?y = 'book')
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>
```

```
SELECT ?book
FROM <Amazon.owl>
WHERE
{
   ?book rdf:type a:Book
}
```

Consulta reescrita oriunda de um processo básico (Q_{BAS})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>

SELECT ?book
FROM <Amazon.owl>
WHERE
{
   ?book rdf:type a:Book .
   ?book rdf:type a:Book .
   FILTER (?y = 'book')
   FILTER (?y= 'book')
}
```

Q_4 (Alvo: *Sales* – Fonte: *Amazon*)

Linguagem Natural

Retorne todos os Produtos que não sejam Livros

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://www.owl-ontologies.com/sales.owl#>

SELECT ?prod
FROM <Sales.owl>
WHERE{
   ?prod rdf:type s:Product
   OPTIONAL {
      ?y rdf:type s:Publication .
      ?y s:type ?k
      FILTER ((?prod = ?y)&&(?k = 'book'))
   }
   FILTER (!BOUND(?y))
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>

SELECT ?prod
FROM <Amazon.owl>
WHERE{
   {
    {?prod rdf:type a:DVD}
    UNION
    {?prod rdf:type a:Music}
    UNION
    {?prod rdf:type a:Book}
   }
   OPTIONAL {
```

```
      ?y rdf:type a:Book
      FILTER ((?prod = ?y))
   }
   FILTER (!BOUND(?y))
}
```

Consulta reescrita oriunda de um processo básico (Q_{BAS})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>

SELECT ?prod
FROM <Amazon.owl>
WHERE{
   {
    {?prod rdf:type a:DVD}
    UNION
    {?prod rdf:type a:Music}
    UNION
    {?prod rdf:type a:Book}
   }
   OPTIONAL {
      ?y rdf:type a:Book .
      ?y rdf:type a:Book
      FILTER ((?prod = ?y)&&(?k = 'book'))
      FILTER (?k= 'book')
   }
   FILTER (!BOUND(?y))
}
```

Q_5 (Alvo: *Sales* – Fonte: *Amazon*)

Linguagem Natural

Retorne todas as Editoras (Opção escolhida: SELECT e CONSTRUCT na reescrita)

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://www.owl-ontologies.com/sales.owl#>

SELECT ?edit
FROM <Sales.owl>
WHERE
{
   ?edit rdf:type s:Publisher
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>

SELECT ?edit
FROM <Amazon.owl>
WHERE {
   ?_x5 a:publisher ?_y5 .
   ?_x5 rdf:type a:Book
   LET (?edit := IRI(fn:concat(a:,?_y5)))
}
-----------------------------------------------
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>
PREFIX s:   <http://www.owl-ontologies.com/sales.owl#>

CONSTRUCT {?edit rdf:type s:Publisher}
FROM <Sales.owl>
FROM <Amazon.owl>
WHERE {
   ?_x5 a:publisher ?_y5 .
   ?_x5 rdf:type a:Book
   LET (?edit := IRI(fn:concat(a:,?_y5)))
}
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

```
        ------------------------ Não Reescreve--------------------
```

Q_6 (Alvo: *Sales* – Fonte: *Amazon*)

Linguagem Natural

Retorne o isbn, o título e o autor de todos os livros que custam mais de 30 reais. Opcionalmente, retorne o nome e o endereço da editora, caso estes estejam informados. Ordene os resultados pelo título.

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://www.owl-ontologies.com/sales.owl#>

SELECT ?isbn ?tit ?ned ?eed ?aut
FROM <Sales.owl>
WHERE
{
   ?l s:isbn ?isbn .
   ?l s:title ?tit .
   OPTIONAL
   {
     ?l s:publishedBy ?e .
     ?e s:pubName ?ned .
     ?e s:pubAddress ?eed
   }
   ?l s:author ?aut .
   ?l s:type ?y .
   ?l s:price ?p
   FILTER((?y = 'book') && (?p > 30))
}
ORDER BY(?tit)
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>

SELECT ?isbn ?tit ?ned ?eed ?aut
FROM <Amazon.owl>
WHERE
{
   ?l a:isbn ?isbn .
   ?l a:description ?tit .
   ?l rdf:type a:Book .
   ?l a:author ?aut .
   ?l a:price ?p
```

```
    OPTIONAL
    {
       ?l a:publisher ?ned .
       ?l rdf:type a:Book .
       ?l a:publisherAddress ?eed
    }
    FILTER(?p > 30)
}
ORDER BY(?tit)
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

---------- Não Reescreve ----------

Q_7 (Alvo: *Sales* – Fonte: *Amazon*)

Linguagem Natural

Retorne todas as Músicas ou todos os Vídeos, juntamente com os títulos destes.

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://www.owl-ontologies.com/sales.owl#>

SELECT ?prod ?tit
FROM <Sales.owl>
WHERE
{
    {
     {?prod rdf:type s:Music}
     UNION
     {?prod rdf:type s:Video}
    }
    .
    {?prod s:title ?tit}
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>

SELECT ?prod ?tit
FROM <Amazon.owl>
WHERE
{
    {?prod rdf:type a:Music . ?prod a:description ?tit}
    UNION
    {?prod rdf:type a:DVD . ?prod a:description ?tit}
}
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>

SELECT ?prod ?tit
FROM <Amazon.owl>
WHERE
{
    {
     {?prod rdf:type a:Music}
     UNION
```

```
     {?prod rdf:type a:DVD}
    }
    .
    {
     {?prod a:description ?tit . ?prod rdf:type a:Book}
     UNION
     {?prod a:description ?tit . ?prod rdf:type a:DVD}
     UNION
     {?prod a:description ?tit . ?prod rdf:type a:Music}
    }
}
```

Q_8 (Alvo: *Sales* – Fonte: *Amazon*)

Linguagem Natural

Retorne o título de todos os Vídeos, juntamente com o nome do diretor. Se o gênero estiver informado, retorne-o também.

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://www.owl-ontologies.com/sales.owl#>

SELECT ?tit ?dir ?gen
FROM <Sales.owl>
WHERE
{
   ?vid s:title ?tit .
   ?vid s:director ?dir .
   ?vid rdf:type s:Video
   OPTIONAL{?vid s:genre ?gen}
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>

SELECT ?tit ?dir ?gen
FROM <Amazon.owl>
WHERE
{
   ?vid  rdf:type a:DVD .
   ?vid  a:description ?tit .
   ?vid  a:hasDirector ?_z16 .
   ?_z16 a:directorName ?dir
   OPTIONAL {?vid a:category ?gen . ?vid rdf:type a:DVD}
}
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>

SELECT ?tit ?dir ?gen
FROM <Amazon.owl>
WHERE
{
   {
    {?vid a:description ?tit . ?vid rdf:type a:Book}
    UNION
```

```
     {?vid a:description ?tit . ?vid rdf:type a:DVD}
     UNION
     {?vid a:description ?tit . ?vid rdf:type a:Music}
    }
    .
    ?vid  a:hasDirector ?_z16 .
    ?_z16 a:directorName ?dir .
    ?vid rdf:type a:DVD .
    ?vid rdf:type a:DVD
    OPTIONAL {?vid a:category ?gen . ?vid rdf:type a:DVD}
}
```

Q_9 (Alvo: *Sales* – Fonte: *Amazon*)

Linguagem Natural

Retorne o título de todas as Músicas, com seus respectivos intérpretes.

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://www.owl-ontologies.com/sales.owl#>

SELECT ?tit ?nin
FROM <Sales.owl>
WHERE
{
   ?mus s:title ?tit .
   ?mus s:artist ?nin
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>

SELECT ?tit ?nin
FROM <Amazon.owl>
WHERE
{
   ?mus  a:description ?tit .
   ?mus rdf:type a:Music .
   ?mus  a:artistName ?nin
}
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>

SELECT ?tit ?nin
FROM <Amazon.owl>
WHERE
{
   {
    {?mus a:description ?tit . ?mus rdf:type a:Book}
    UNION
    {?mus a:description ?tit . ?mus rdf:type a:DVD}
    UNION
    {?mus a:description ?tit . ?mus rdf:type a:Music}
   }
   .
```

```
   ?mus  a:artistName ?nin .
   ?mus rdf:type a:Music
}
```

Q_{10} (Alvo: *Sales* – Fonte: *Amazon*)

Linguagem Natural

Retorne todas as Gravadoras para as quais Caetano Veloso gravou alguma Música.

(Opção escolhida: SELECT e CONSTRUCT na reescrita)

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://www.owl-ontologies.com/sales.owl#>

SELECT ?grv
FROM <Sales.owl>
WHERE
{
   ?grv rdf:type s:RecordLabel .
   ?mus s:rec ?grv .
   ?mus s:artist ?y
   FILTER(?y = 'Caetano Veloso')
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>

SELECT ?grv
FROM <Amazon.owl>
WHERE
{
   ?mus a:producedBy ?_z6 .
   ?_z6 a:recLabel ?_y6 .
   ?mus rdf:type a:Music .
   ?mus a:artistName ?y
   FILTER(?y = 'Caetano Veloso')
   LET (?grv := IRI(fn:concat(a:,?_y6)))
}
-----------------------------------------------
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>
PREFIX s:   <http://www.owl-ontologies.com/sales.owl#>

CONSTRUCT {?grv rdf:type s:RecordLabel . ?mus s:rec ?fRLabel}
FROM <Sales.owl>
FROM <Amazon.owl>
WHERE
{
   ?mus a:producedBy ?_z6 .
   ?_z6 a:recLabel ?_y6 .
   ?mus rdf:type a:Music .
   ?mus a:artistName ?y
   FILTER(?y = 'Caetano Veloso')
   LET (?grv := IRI(fn:concat(a:,?_y6)))
}
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

----------Não Reescreve----------

Q_{11} (Alvo: *Sales* – Fonte: *Ebay*)

Linguagem Natural

Retorne todas as Músicas.

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://www.owl-ontologies.com/sales.owl#>

SELECT ?mus
FROM <Sales.owl>
WHERE
{
   ?mus rdf:type s:Music
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX e:   <http://www.owl-ontologies.com/Ebay#>

SELECT ?mus
FROM <Ebay.owl>
WHERE
{
   ?mus rdf:type e:Product .
   ?mus e:type ?_k1
   FILTER(?_k1 = 'music')
}
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX e:   <http://www.owl-ontologies.com/Ebay#>

SELECT ?mus
FROM <Ebay.owl>
WHERE
{
   ?mus rdf:type e:Product .
   ?mus e:type ?_k1
   FILTER(?_k1 = 'music')
}
```

Q_{12} (Alvo: *Ebay* – Fonte: Amazon)

Linguagem Natural

Retorne todos os Livros.

SPARQL

```
PREFIX e:   <http://www.owl-ontologies.com/Ebay#>

SELECT ?prod
FROM <Ebay.owl>
WHERE
{
   ?prod e:type ?y
   FILTER(?y = 'book')
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>

SELECT ?prod
FROM <Amazon.owl>
WHERE
{
   ?prod rdf:type a:Book
}
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>

SELECT ?prod
FROM <Amazon.owl>
WHERE
{
   {
    ?prod rdf:type a:Book
    FILTER(?y = 'book')
   }
   UNION
   {
    ?prod rdf:type a:Music
    FILTER(?y = 'music')
   }
   FILTER(?y = 'book')
}
```

Q_{13} (Alvo: *Ebay* – Fonte: Amazon)

Linguagem Natural

Retorne a descrição de todas as Músicas.

SPARQL

```
PREFIX e:   <http://www.owl-ontologies.com/Ebay#>

SELECT ?dsc
FROM <Ebay.owl>
WHERE
{
   ?prod e:description ?dsc .
   ?prod e:type ?y
   FILTER(?y = 'music')
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>

SELECT ?dsc
FROM <Amazon.owl>
WHERE
{
   ?prod a:description ?dsc .
   ?prod rdf:type a:Music
}
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX a:   <http://www.owl-ontologies.com/Amazon.owl#>

SELECT ?dsc
FROM <Amazon.owl>
WHERE
{
   {
    {?prod a:description ?dsc . ?prod rdf:type a:Book}
    UNION
    {?prod a:description ?dsc . ?prod rdf:type a:Music}
   }
   .
   {
    {?prod rdf:type a:Book FILTER(?y = 'book')}
    UNION
    {?prod rdf:type a:Music FILTER(?y = 'music')}
   }
   FILTER(?y = 'music')
}
```

Apêndice G
Consultas Testadas – Ontologias do Domínio de Educação

Q_{14} (Alvo: *Semiport* – Fonte: *UnivBench*)

Linguagem Natural

Retorne o nome e todos que fazem algum curso, juntamente com o nome do curso

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://swrc.ontoware.org/ontology/portal#>

SELECT ?n ?cn
FROM <Semiport.owl>
WHERE
{
  ?p s:name ?n .
  ?p s:course ?cn
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX u:   <http://www.lehigh.edu/~zhp2/univ-bench.owl#>

SELECT ?p ?cn
FROM <UnivBench.owl>
WHERE
{
  ?p u:personName ?n . ?p rdf:type u:UndergraduateStudent .
  ?p u:takesCourse ?_z6 . ?z6 u:courseName ?cn
}
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX u:   <http://www.lehigh.edu/~zhp2/univ-bench.owl#>

SELECT ?p ?cn
FROM <UnivBench.owl>
WHERE
{
  {
   {?p u:personName ?n . ?p rdf:type u:AssistantProfessor}
   UNION
   {?p u:personName ?n . ?p rdf:type  u:FullProfessor}
   UNION
   {?p u:personName ?n . ?p rdf:type  u:AdministrativeStaff}
   UNION
   {?p u:personName ?n . ?p rdf:type  u:Student}
   UNION
   {?p u:personName ?n . ?p rdf:type  u:GraduateStudent}
   UNION
   {?p u:personName ?n . ?p rdf:type  u:UndergraduateStudent}
  }
  .
  {
    ?p u:takesCourse ?_z6 . ?z6 u:courseName ?cn .
    ?p rdf:type u:UndergraduateStudent
```

```
   }
}
```

Q_{15} (Alvo: *Semiport* – Fonte: *UnivBench*)

Linguagem Natural

Retorne todos os líderes de algum grupo de pesquisa

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://swrc.ontoware.org/ontology/portal#>

SELECT ?p
FROM <Semiport.owl>
WHERE
{
  ?p s:headOfGroup ?g
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX u:   <http://www.lehigh.edu/~zhp2/univ-bench.owl#>

SELECT ?p ?cn
FROM <UnivBench.owl>
WHERE
{
   {?p u:headOfOrganization ?g . ?p rdf:type u:AssistantProfessor}
   UNION
   {?p u:headOfOrganization ?g . ?p rdf:type  u:FullProfessor}
   UNION
   {?p u:headOfOrganization ?g . ?p rdf:type  u:AdministrativeStaff}
}
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX u:   <http://www.lehigh.edu/~zhp2/univ-bench.owl#>

SELECT ?p ?cn
FROM <UnivBench.owl>
WHERE
{
   {?p u:headOfOrganization ?g . ?p rdf:type u:AssistantProfessor}
   UNION
   {?p u:headOfOrganization ?g . ?p rdf:type  u:FullProfessor}
   UNION
   {?p u:headOfOrganization ?g . ?p rdf:type  u:AdministrativeStaff}
}
```

Q_{16} (Alvo: *Semiport* – Fonte: *UnivBench*)

Linguagem Natural

Retorne todas as organizações que possuem algum membro

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://swrc.ontoware.org/ontology/portal#>
```

```
SELECT ?org
FROM <Semiport.owl>
WHERE{
   ?org rdf:type s:Organization .
   ?org s:member ?p
   FILTER (BOUND(?p))
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX u:   <http://www.lehigh.edu/~zhp2/univ-bench.owl#>

SELECT ?org
FROM <UnivBench.owl>
WHERE
{
  {?org rdf:type u:Department . ?org u:member ?p}
  UNION
  {?org rdf:type u:ResearchGroup . ?org u:member ?p}
  UNION
  {?org rdf:type u:University . ?org u:member ?p}
  FILTER (BOUND(?p))
}
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX u:   <http://www.lehigh.edu/~zhp2/univ-bench.owl#>

SELECT ?org
FROM <UnivBench.owl>
WHERE{
   {
    {?org rdf:type u:Department}
    UNION
    {?org rdf:type u:ResearchGroup}
    UNION
    {?org rdf:type u:University}
   }
   .
   {
    {?org u:member ?p . ?org rdf:type u:Department}
    UNION
    {?org u:member ?p . ?org rdf:type u:ResearchGroup}
    UNION
    {?org u:member ?p . ?org rdf:type u:University}
   }
   FILTER (BOUND(?p))
}
```

Q_{17} (Alvo: *Semiport* – Fonte: *UnivBench*)

Linguagem Natural

Retorne todas as organizações envolvidas em projetos

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://swrc.ontoware.org/ontology/portal#>
```

```
SELECT ?org
FROM <Semiport.owl>
WHERE{
   ?org rdf:type s:Organization .
   ?org s:carriesOut ?proj
   FILTER(BOUND(?proj))
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX u:   <http://www.lehigh.edu/~zhp2/univ-bench.owl#>

SELECT ?org
FROM <UnivBench.owl>
WHERE{
   ?org rdf:type u:ResearchGroup.
   ?org u:researchProject ?proj
   FILTER(BOUND(?proj))
}
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX u:   <http://www.lehigh.edu/~zhp2/univ-bench.owl#>

SELECT ?org
FROM <UnivBench.owl>
WHERE{
   {
    {?org rdf:type u:Department}
    UNION
    {?org rdf:type u:ResearchGroup}
    UNION
    {?org rdf:type u:University}
   }
   .
   {
     ?org u:researchProject ?proj .
     ?org rdf:type u:ResearchGroup
   }
   FILTER (BOUND(?proj))
}
```

Q_{18} (Alvo: *Semiport* – Fonte: *UnivBench*)

Linguagem Natural

Retorne o nome de todos os cursos

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://swrc.ontoware.org/ontology/portal#>

SELECT ?nCourse
FROM <Semiport.owl>
WHERE{
  ?stu s:course ?nCourse
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
PREFIX u:   <http://www.lehigh.edu/~zhp2/univ-bench.owl#>

SELECT ?nCourse
FROM <UnivBench.owl>
WHERE{
  ?stu  u:takesCourse ?_z26 .
  ?_z26 u:courseName ?nCourse .
  ?stu rdf:type u:UndergraduateStudent
}
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX u:   <http://www.lehigh.edu/~zhp2/univ-bench.owl#>

SELECT ?nCourse
FROM <UnivBench.owl>
WHERE{
  ?stu  u:takesCourse ?_z26 .
  ?_z26 u:courseName ?nCourse .
  ?stu rdf:type u:UndergraduateStudent
}
```

Q_{19} (Alvo: *UnivBench* – Fonte: *Semiport*)

Linguagem Natural

Retorne os autores da publicação cujo título é "*Linking Data to Ontologies*"

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX u:   <http://www.lehigh.edu/~zhp2/univ-bench.owl#>

SELECT ?aut
FROM <UnivBench.owl>
WHERE{
  ?pub u:publicationAuthor ?aut .
  ?pub u:pubTitle ?tit .
  FILTER(?tit = 'Linking Data to Ontologies')
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://swrc.ontoware.org/ontology/portal#>

SELECT ?aut
FROM <Semiport.owl>
WHERE
{
     {?pub s:publicationAuthor ?aut . ?pub rdf:type s:Article .
      ?pub s:pubTitle ?tit}
    UNION
     {?pub s:publicationAuthor ?aut . ?pub rdf:type s:Book .
      ?pub s:pubTitle ?tit}
    UNION
     {?pub s:publicationAuthor ?aut . ?pub rdf:type s:Manual .
      ?pub s:pubTitle ?tit}
    UNION
     {?pub s:publicationAuthor ?aut . ?pub rdf:type s:TechnicalReport
.
      ?pub s:pubTitle ?tit}
```

```
    UNION
     {?pub s:publicationAuthor ?aut . ?pub rdf:type
s:UnofficialPublication .
      ?pub s:pubTitle ?tit}
    FILTER(?tit = 'Linking Data to Ontologies')
}
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://swrc.ontoware.org/ontology/portal#>

SELECT ?aut
FROM <Semiport.owl>
WHERE{
   {
     {?pub s:publicationAuthor ?aut . ?pub rdf:type s:Article}
    UNION
     {?pub s:publicationAuthor ?aut . ?pub rdf:type s:Book}
    UNION
     {?pub s:publicationAuthor ?aut . ?pub rdf:type s:Manual}
    UNION
     {?pub s:publicationAuthor ?aut . ?pub rdf:type s:TechnicalReport}
    UNION
     {?pub s:publicationAuthor ?aut . ?pub rdf:type
s:UnofficialPublication}
   }
   .
   {
     {?pub s:pubTitle ?tit . ?pub rdf:type s:Article}
    UNION
     {?pub s:pubTitle ?tit . ?pub rdf:type s:Book}
    UNION
     {?pub s:pubTitle ?tit . ?pub rdf:type s:Manual}
    UNION
     {?pub s:pubTitle ?tit . ?pub rdf:type s:TechnicalReport}
    UNION
     {?pub s:pubTitle ?tit . ?pub rdf:type s:UnofficialPublication}
   }
   FILTER(?tit = 'Linking Data to Ontologies')
}
```

Q_{20} (Alvo: *UnivBench* – Fonte: *Semiport*)

Linguagem Natural

Retorne o nome de todos os cursos. Ordene pelo nome.

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX u:   <http://www.lehigh.edu/~zhp2/univ-bench.owl#>

SELECT ?cName
FROM <UnivBench.owl>
WHERE{
  ?c u:courseName ?cName
}ORDER BY ?cName
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://swrc.ontoware.org/ontology/portal#>
```

```
SELECT ?cName
FROM <Semiport.owl>
WHERE
{
  ?_x28 s:course ?cName . ?_x28 rdf:type s:UndergraduateStudent
} ORDER BY ?cName
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

----------Não Reescreve----------

Q_{21} (Alvo: *UnivBench* – Fonte: *Semiport*)

Linguagem Natural

Retorne o nome de todos que fazem algum curso, juntamente com o curso (SELECT e CONSTRUCT)

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX u:   <http://www.lehigh.edu/~zhp2/univ-bench.owl#>

SELECT ?n ?course
FROM <UnivBench.owl>
WHERE{
  ?p u:personName ?n .
  ?p u:takesCourse ?course
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://swrc.ontoware.org/ontology/portal#>

SELECT ?n ?course
FROM <Semiport.owl>
WHERE
{
 ?p s:name ?n . ?p rdf:type s:UndergraduateStudent . ?p s:course ?_y29
 LET (?course := IRI(fn:concat(s:,?_y29)))
}
------------------------------------------------------------
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://swrc.ontoware.org/ontology/portal#>
PREFIX u:   <http://www.lehigh.edu/~zhp2/univ-bench.owl#>

CONSTRUCT {?p u:personName ?n . ?p u:takesCourse ?course}
FROM <Semiport.owl>
FROM <UnivBench.owl>
WHERE
{
 ?p s:name ?n . ?p rdf:type s:UndergraduateStudent . ?p s:course ?_y29
 LET (course := IRI(fn:concat(s:,?_y29)))
}
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

----------Não Reescreve----------

Q_{22} (Alvo: *UnivBench* – Fonte: *Semiport*)

Linguagem Natural

Retorne todas as organizações que são suborganizações de outras.

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX u:   <http://www.lehigh.edu/~zhp2/univ-bench.owl#>

SELECT ?org1
FROM <UnivBench.owl>
WHERE
{ ?org1 u:subOrganizationOf ?org2 }
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://swrc.ontoware.org/ontology/portal#>

SELECT distinct ?org1
FROM <Semiport.owl>
WHERE
{?org1 s:isOf ?org2 . ?org1 rdf:type s:Department}
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://swrc.ontoware.org/ontology/portal#>

SELECT distinct ?org1
FROM <Semiport.owl>
WHERE
{?org1 s:isOf ?org2 . ?org1 rdf:type s:Department}
```

Q_{23} (Alvo: *UnivBench* – Fonte: *Semiport*)

Linguagem Natural

Retorne todas as organizações que não são suborganizações de outras.

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX u:   <http://www.lehigh.edu/~zhp2/univ-bench.owl#>

SELECT ?org1
FROM <UnivBench.owl>
WHERE
{ ?org1 rdf:type u:Organization
  OPTIONAL
  {
   ?orgSub u:subOrganizationOf ?org2
   FILTER(?org1 = ?orgSub)
  }
  FILTER(!BOUND(?orgSub))
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://swrc.ontoware.org/ontology/portal#>
```

```
SELECT ?org1
FROM <Semiport.owl>
WHERE
{ {
     {?org1 rdf:type s:Organization}
    UNION
     {?org1 rdf:type s:Department}
    UNION
     {?org1 rdf:type s:ResearchGroup}
    UNION
     {?org1 rdf:type s:University}
  }
  OPTIONAL
  {
   ?orgSub s:isOf ?org2 . ?orgSub rdf:type s:Departament
   FILTER(?org1 = ?orgSub)
  }
  FILTER(!BOUND(?orgSub))
}
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://swrc.ontoware.org/ontology/portal#>

SELECT ?org1
FROM <Semiport.owl>
WHERE
{
  {
     {?org1 rdf:type s:Organization}
    UNION
     {?org1 rdf:type s:Department}
    UNION
     {?org1 rdf:type s:ResearchGroup}
    UNION
     {?org1 rdf:type s:University}
  }
  OPTIONAL
  {
   ?orgSub s:isOf ?org2 . ?orgSub rdf:type s:Departament
   FILTER(?org1 = ?orgSub)
  }
  FILTER(!BOUND(?orgSub))
}
```

Q_{24} (Alvo: *UnivBench* – Fonte: *Semiport*)

Linguagem Natural

Retorne todos os professores assistentes que orientam alunos de pós-graduação. Opcionalmente, retorne email deste professores.

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX u:   <http://www.lehigh.edu/~zhp2/univ-bench.owl#>

SELECT ?prof ?email
FROM <UnivBench.owl>
WHERE
```

```
{
  ?prof rdf:type u:AssistantProfessor .
  ?prof u:advices ?stu
  OPTIONAL{ ?prof u:emailAddress ?email}
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

O algoritmo detectará que a consulta retornará vazio – não prossegue a reescrita

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://swrc.ontoware.org/ontology/portal#>

SELECT ?prof ?email
FROM <Semiport.owl>
WHERE
{
  {?prof rdf:type s:AssistantProfessor} .
  {?prof s:advises ?stu . ?prof rdf:type s:FullProfessor}
  OPTIONAL
  {
    {?prof s:emailAddress ?email . ?prof rdf:type
s:AssistantProfessor}
     UNION
    {?prof s:emailAddress ?email . ?prof rdf:type s:FullProfessor}
     UNION
    {?prof s:emailAddress ?email . ?prof rdf:type
s:AdministrativeStaff}
     UNION
    {?prof s:emailAddress ?email . ?prof rdf:type s:Student}
     UNION
    {?prof s:emailAddress ?email . ?prof rdf:type s:GraduateStudent}
     UNION
    {?prof s:emailAddress ?email . ?prof rdf:type
s:UndergraduateStudent}
  }
}
```

Q_{25} (Alvo: *UnivBench* – Fonte: *Semiport*)

Linguagem Natural

Retorne todos os professores. Opcionalmente, retorne o email dos professores assistentes que orientam alunos de pós-graduação.

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX u:   <http://www.lehigh.edu/~zhp2/univ-bench.owl#>

SELECT ?prof ?email
FROM <UnivBench.owl>
WHERE
{
  ?prof rdf:type u:Faculty
  OPTIONAL
  {
  ?prof u:emailAddress ?email .
```

```
  ?prof rdf:type u:AssistantProfessor .
  ?prof u:advices ?stu
  }
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://swrc.ontoware.org/ontology/portal#>

SELECT ?prof
FROM <Semiport.owl>
WHERE
{
   {?prof rdf:type s:AssistantProfessor}
   UNION
   {?prof rdf:type s:FullProfessor}
}
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://swrc.ontoware.org/ontology/portal#>

SELECT ?prof
FROM <Semiport.owl>
WHERE
{
 { {?prof rdf:type s:AssistantProfessor}
   UNION
   {?prof rdf:type s:FullProfessor}}
 OPTIONAL
 {
   {
    {?prof s:emailAddress ?email . ?prof rdf:type
s:AssistantProfessor}
     UNION
    {?prof s:emailAddress ?email . ?prof rdf:type s:FullProfessor}
     UNION
    {?prof s:emailAddress ?email . ?prof rdf:type
s:AdministrativeStaff}
     UNION
    {?prof s:emailAddress ?email . ?prof rdf:type s:Student}
     UNION
    {?prof s:emailAddress ?email . ?prof rdf:type s:GraduateStudent}
     UNION
    {?prof s:emailAddress ?email . ?prof rdf:type
s:UndergraduateStudent}
   }
   .
   {?prof rdf:type s:AssistantProfessor}
   .
   {?prof s:advises ?stu . ?prof rdf:type s:FullProfessor}
 }
}
```

Q_{26} (Alvo: *UnivBench* – Fonte: *Semiport*)

Linguagem Natural

Retorne todos os estudantes do curso cujo nome é Computação ou todos os professores

SPARQL

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
PREFIX u:   <http://www.lehigh.edu/~zhp2/univ-bench.owl#>

SELECT ?p
FROM <UnivBench.owl>
WHERE
{
  {?p rdf:type u:Student .
   ?p u:takesCourse ?c .
   ?c u:courseName "Computacao"}
  UNION
  {?p rdf:type u:Faculty}
}
```

Reescrita automática utilizando a *SQuOL* (Q_{SQU})

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s:   <http://swrc.ontoware.org/ontology/portal#>

SELECT distinct ?p
FROM <Semiport.owl>
WHERE
{
 {
  ?p rdf:type s:UndergraduateStudent . ?p s:course "Computacao"
 }
 UNION
 {
  {?p rdf:type s:AssistantProfessor}
  UNION
  {?p rdf:type s:FullProfessor }
 }
}
```

Consulta reescrita oriunda de um processo básico *(Q_{BAS})*

```
-----------------------Não Reescreve----------------------
```