UNIVERSIDADE FEDERAL DO CEARÁ
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

REGIS PIRES MAGALHÃES

UM AMBIENTE PARA PROCESSAMENTO DE CONSULTAS FEDERADAS EM LINKED DATA MASHUPS

FORTALEZA, CEARÁ

2012

REGIS PIRES MAGALHÃES

UM AMBIENTE PARA PROCESSAMENTO DE CONSULTAS FEDERADAS EM LINKED DATA MASHUPS

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.

Área de concentração: Banco de Dados

Orientador: Profa. Dra. Vânia Maria Ponte Vidal

Coorientador: Prof. Dr. José Antonio F. de Macedo

FORTALEZA, CEARÁ

2012

A000z MAGALHÃES, REGIS P.

Um ambiente para processamento de consultas federadas em Linked Data Mashups / Regis Pires Magalhães. 2012.

117p.;il. color. enc.

Orientador: Profa. Dra. Vânia Maria Ponte Vidal

Co-Orientador: Prof. Dr. José Antonio F. de Macedo

Dissertação(Ciência da Computação) - Universidade Federal do Ceará, Departamento de Computação, Fortaleza, 2012.

1. Consultas Federadas 2. Integração de dados 3. Linked Data Mashups I. Profa. Dra. Vânia Maria Ponte Vidal(Orient.) II. Universidade Federal do Ceará– Ciência da Computação(Mestrado) III. Mestre

CDD:000.0

REGIS PIRES MAGALHÃES

UM AMBIENTE PARA PROCESSAMENTO DE CONSULTAS FEDERADAS EM LINKED DATA MASHUPS

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. Área de concentração: Banco de Dados

Aprovada em: __/__/____

BANCA EXAMINADORA

Profa. Dra. Vânia Maria Ponte Vidal
Universidade Federal do Ceará - UFC
Orientador

Prof. Dr. José Antonio F. de Macedo
Universidade Federal do Ceará - UFC
Coorientador

Prof. Dr. José Maria da Silva Monteiro Filho
Universidade Federal do Ceará - UFC

Prof. Dr. Fábio André Machado Porto
Laboratório Nacional de Computação
Científica - LNCC

A todos aqueles que efetivamente contribuem para tornar o mundo melhor.

AGRADECIMENTOS

Agradeço a Deus e aos meus pais por existir e poder aprender e compartilhar o que aprendo a cada dia.

Agradeço também ao apoio e incentivo constante de minha esposa Evelane, dos meus pais Flávio e Linda, bem como dos meus irmãos Mario e Emanuelle.

Aos meus maravilhosos filhos Renato e Luana Stanz, agradeço pela compreensão durante todos os momentos em que não pudemos estar juntos devido às atividades de pesquisa desenvolvidas no decorrer do mestrado.

Aos Professores Vânia Vidal e José Antônio Macêdo pela orientação e coorientação, respectivamente, bem como por todo o apoio recebido durante todo o decorrer da pesquisa. Ao Professor José Maria Monteiro pelas conversas, ideias e incentivos constantes.

Ao Professor Fábio Porto que sempre foi muito atencioso e preciso em suas observações que foram essenciais para que este trabalho pudesse ser realizado. Felizmente pudemos conviver e conversar bastante no LNCC, através do Skype e em outros momentos. Agradeço também a sua acolhedora equipe do Laboratório DEXL e à Professora Ana Maria Moura que tão bem me recepcionaram no LNCC.

Aos Professores Marco A. Casanova e Karin Breitman do Departamento de Informática da PUC-Rio que me propiciaram a grata oportunidade de participar de seus grupos de pesquisa durante minha missão de estudos na PUC. Também foram bastante importantes as parcerias de pesquisa e as amizades que iniciamos com os pesquisadores Percy Salas, Edgard Marx e Jaumir Silveira, também da PUC-Rio.

Ao amigo João Pinheiro pelos vários esclarecimentos práticos sobre Web Semântica e Linked Data que foram de grande utilidade para o desenvolvimento deste trabalho.

Ao amigo de mestrado Macêdo Maia pelas valiosas contribuições relacionadas aos experimentos realizados para este trabalho e pelos aprendizados que pudemos fazer juntos sobre Linked Data.

Ao amigo Franzé Júnior pela grande ajuda durante a implementação do ambiente Web para a execução de consultas federadas. Ao bolsista de iniciação científica Dener da Silva Miranda por também ter contribuído.

Aos amigos de mestrado e doutorado Diego Victor, Mônica Regina, Manoel Siqueira, Diego Sá, Carlos Eduardo (Cadu), Arlino Henrique, Luís Eufrásio, Igo Brilhante, Henrique Viana, Ticiana Linhares, Lívia Almada, Vinícius Pires, Flávio Sousa, Leonardo Moreira, Paulo Rego, Bruno Leal, Ticianne Ribeiro, David Araújo, Jeovane Reges, Josefran Bastos e Francicleber Ferreira pelos excelentes momentos de convívio e aprendizado que pudemos compartilhar ao longo dessa caminhada.

A todos aqueles que não foram citados até aqui, mas que também contribuíram direta ou indiretamente para a realização deste trabalho.

Ao apoio recebido pelo Instituto Federal do Piauí (IFPI) por permitir meu afastamento das atividades docentes para cursar o mestrado durante os anos de 2010 e 2011. Ao apoio financeiro recebido da CAPES.

“Se você tem uma maçã e eu tenho uma maçã e nós trocamos essas maçãs, então eu e você ainda teremos uma maçã cada. Mas se você tiver uma idéia e eu tiver uma idéia e nós trocamos idéias, então cada um de nós terá duas idéias.”

(George Bernard Shaw)

RESUMO

Tecnologias da Web Semântica como modelo RDF, URIs e linguagem de consulta SPARQL, podem reduzir a complexidade de integração de dados ao fazer uso de ligações corretamente estabelecidas e descritas entre fontes. No entanto, a dificuldade para formulação de consultas distribuídas tem sido um obstáculo para aproveitar o potencial dessas tecnologias em virtude da autonomia, distribuição e vocabulário heterogêneo das fontes de dados. Esse cenário demanda mecanismos eficientes para integração de dados sobre Linked Data. Linked Data Mashups permitem aos usuários executar consultas e integrar dados estruturados e vinculados na web. O presente trabalho propõe duas arquiteturas de Linked Data Mashups: uma delas baseada no uso de mediadores e a outra baseada no uso de Linked Data Mashup Services (LIDMS). Um módulo para execução eficiente de planos de consulta federados sobre Linked Data foi desenvolvido e é um componente comum a ambas as arquiteturas propostas. A viabilidade do módulo de execução foi demonstrada através de experimentos. Além disso, um ambiente Web para execução de LIDMS também foi definido e implementado como contribuições deste trabalho.

Palavras-chave: Consultas Federadas. Integração de dados. Linked Data Mashups.

ABSTRACT

Semantic Web technologies like RDF model, URIs and SPARQL query language, can reduce the complexity of data integration by making use of properly established and described links between sources. However, the difficulty to formulate distributed queries has been a challenge to harness the potential of these technologies due to autonomy, distribution and vocabulary of heterogeneous data sources. This scenario demands effective mechanisms for integrating data on Linked Data. Linked Data Mashups allow users to query and integrate structured and linked data on the web. This work proposes two architectures of Linked Data Mashups: one based on the use of mediators and the other based on the use of Linked Data Mashup Services (LIDMS). A module for efficient execution of federated query plans on Linked Data has been developed and is a component common to both proposed architectures. The execution module feasibility has been demonstrated through experiments. Furthermore, a LIDMS execution Web environment also has been defined and implemented as contributions of this work.

Keywords: Federated Queries. Data Integration. Linked Data Mashups.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A Web é atualmente um enorme espaço global de documentos e dados distribuídos em múltiplas fontes heterogêneas. Esta nova Web, denominada Web de Dados, visa pavimentar o caminho para a Web Semântica funcional, onde haverá a disponibilidade de uma grande quantidade de dados vinculados no modelo RDF.

A Web Semântica provê tecnologias para publicação, recuperação e integração de dados distribuídos na Web de dados. Resumidamente essas tecnologias são: **RDF** (Resource Description Framework), um modelo de dados simples, expressivo, extensível e que permite interligar itens entre diferentes fontes de dados; **URI (ou IRI)**, usado como mecanismo de nome global; **linguagem SPARQL** (PRUD'HOMMEAUX; SEABORNE, 2008), uma linguagem de consulta de alto nível capaz de abstrair detalhes da sequência de passos necessária para a execução de consultas sobre fontes heterogêneas; e possibilidade de uso de **mecanismos de inferência** sobre os dados. A integração de dados em grande escala é provavelmente uma das melhores aplicações para essas tecnologias, tendo em vista que o volume de dados e a quantidade de conjuntos de dados disponíveis se expandem a cada dia, dificultando cada vez mais seu consumo de forma integrada. Além disso, um conjunto de melhores práticas para publicação e conexão de dados estruturados na Web, conhecido como *Linked Data*, possibilita a interligação de itens entre diferentes fontes de dados e, portanto, a conexão dessas fontes em um único espaço de dados global (HEATH; BIZER, 2011). *Linked Data* se baseia em tecnologias da Web Semântica, e permite reduzir a complexidade de integração de dados devido às ligações estabelecidas e descritas entre os conjuntos de dados. Além disso, a adoção de um modelo de dados padronizado (modelo RDF) e de um mecanismo padronizado de acesso aos dados, juntamente com a natureza autodescritiva dos dados também simplificam sua integração. A arquitetura aberta de *Linked Data* ainda permite a descoberta de novos dados e mesmo conjuntos de dados em tempo de execução. Desse modo, *Linked Data* tem o potencial de facilitar o acesso aos dados semanticamente relacionados, estabelecendo conexões explícitas entre conjuntos de dados distintos a fim de facilitar sua integração e fornecendo, portanto, um novo cenário à integração de dados.

Linked Data Mashups (LDM) são aplicações construídas para realizar integração de dados no contexto de Linked Data. As arquiteturas de aplicações de LDM são muito diversas e dependem largamente do caso de uso (HEATH; BIZER, 2011). Porém, em geral, podemos distinguir duas arquiteturas padrões (HARTIG; ZHAO, 2010), que são representadas na Figura 1.1 e descritas a seguir.

1. **Data warehousing (Enfoque materializado)**. Nesta abordagem os dados são coletados e armazenados em um banco de dados central. Consultas são executadas sobre esse banco de dados. A ideia de *datawarehouse* pode ser aplicada a um serviço de consulta sobre uma coleção de *Linked Data* copiados a partir de múltiplas fontes na Web. Essa coleção de dados pode ser construída através do carregamento de *dumps* dos dados em formato RDF para um *RDF Store*. Também é possível extrair esses dados ao percorrer e analisar informações sobre URIs na Web de Dados, ou através de consultas a um Endpoint SPARQL.

A principal desvantagem dessa abordagem é a replicação dos dados, que além de requerer espaço de armazenamento adicional, ainda possibilita o uso de dados desatualizados em relação aos dados originais. A possível desatualização dos dados leva à necessidade de percorrer a web de dados com certa frequência para atualizá-los. O principal desafio dessa abordagem é, portanto, criar e manter a visão de integração.

2. **Enfoque Virtual**. O enfoque virtual permite a execução de consultas federadas sobre um conjunto fixo de fontes de dados. Seu principal desafio consiste em encontrar planos de execução com desempenho satisfatório sobre múltiplas fontes de dados. A **federação de consultas** baseia-se na distribuição do processamento de consultas para múltiplas fontes de dados autônomas. A visão virtual de integração dos dados pode ser construída com ou sem o uso de um mediador, conforme as explicações a seguir:

   - **Uso de mediador**. Um mediador permite a execução de consultas *ad-hoc* sobre os conjuntos de dados registrados. Um mediador decompõe a consulta em subconsultas de forma transparente, direciona as subconsultas a múltiplos serviços de consulta distribuídos, e, finalmente, integra os resultados das subconsultas. O uso de um mediador aumenta a complexidade da arquitetura. No entanto, dá maior flexibilidade para formulação de novas consultas.
   - **Uso de ambiente para execução de consultas pré-definidas**. O uso de consultas pré-definidas permitem que planos de execução sejam gerados em tempo de projeto. Isso possibilita a análise e otimização das consultas sem as restrições que seriam impostas se essa tarefa ocorresse em tempo de execução. Aplicar a maior otimização possível em tempo de projeto pode viabilizar operações complexas cujo tempo de execução da consulta e transmissão dos resultados dominam o tempo de execução total.

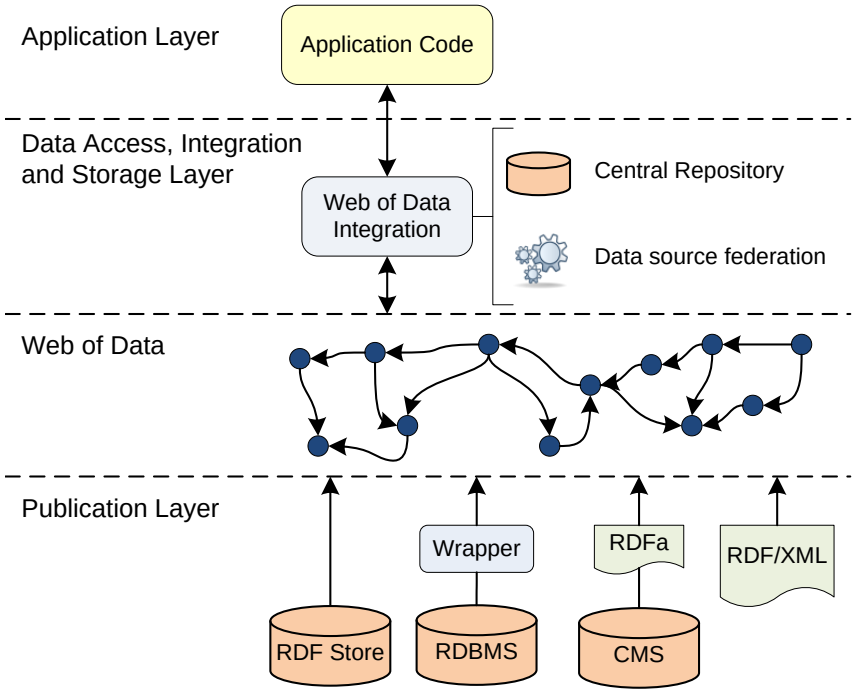

Figura 1.1: Arquitetura Geral de *Linked Data Mashups*

1.1 Motivação

Motivados pelo sucesso da iniciativa *Linking Open Data*[1] e pelo grande crescimento da quantidade de fontes de dados disponíveis na web, novas abordagens de processamento de consultas estão surgindo. Enquanto o processamento de consultas no contexto do modelo RDF era tradicionalmente realizado usando armazenamento centralizado, ultimamente pode-se observar uma mudança de paradigma (SCHWARTE et al., 2011a) para a adoção de abordagens federadas em decorrência da estrutura descentralizada da web. Na prática, inúmeros cenários existem em que mais de uma fonte de dados pode contribuir com informações, tornando o processamento de consultas mais complexo. O caminho natural segue a busca por soluções eficientes para o processamento de consultas federadas sobre fontes de dados distribuídas na web.

1.2 Caracterização do Problema

Abordagens tradicionais de federação de consultas baseiam-se na obtenção de informações úteis para a realização de otimizações, balanceamento de carga ou mesmo uso de dados locais, a partir da cooperação entre as fontes de dados. No entanto, fontes de Linked Data são normalmente publicadas por provedores independentes e autônomos, sobre os quais não temos controle. Essas fontes podem ficar sobrecarregadas ou mesmo inativas em certos momentos não previstos. Além disso, o protocolo SPARQL somente define como a consulta e os resultados são trocados entre clientes e Endpoints, mas não permite a cooperação entre

[1] http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

eles. Realizar consultas federadas eficientes nesse cenário é um grande desafio que vem sendo abordado por algumas estratégias tratadas ao longo deste trabalho. No entanto, as estratégias atuais que endereçam esse problema ainda são relativamente recentes e muitas delas realizam apenas consultas federadas de forma bastante simplificada e sem maiores preocupações com otimização. (GÖRLITZ; STAAB, 2011) afirmam que escalabilidade é o principal aspecto no cenário de federação de consultas, devido ao grande e crescente número de fontes de Linked Data. E isso traz dois grandes desafios: o gerenciamento de estatísticas sobre as fontes de dados, e, otimização e execução eficientes de consultas. Este trabalho lida principalmente com o problema relacionado à execução eficiente de consultas federadas sobre Linked Data.

1.3 Contribuições

A principal contribuição deste trabalho é a implementação e validação, através de experimentos, de um módulo para execução eficiente de planos de consulta federados sobre Linked Data que pode ser usado em arquiteturas de Linked Data Mashups baseadas ou não no uso de mediadores. Este módulo chamado de QEF-LD é uma extensão que adiciona ao QEF – *Query Evaluation Framework* (PORTO et al., 2007) – operadores da álgebra SPARQL, além de outras funcionalidades para acesso e execução eficientes de planos de consulta sobre Linked Data. Detalhes sobre o QEF-LD e sobre os experimentos realizados para validar sua viabilidade serão tratados nos Capítulos 6 e 7, respectivamente.

Outra importante contribuição foi a definição de duas arquiteturas de Linked Data: uma delas baseada no uso de mediadores e a outra baseada no uso de Linked Data Mashup Services (LIDMS), que resumidamente podem ser conceituados como serviços Web capazes de obter dados integrados de fontes Linked Data, a partir da requisição de uma URI. A arquitetura baseada no uso de mediadores será discutida no Capítulo 4 e a arquitetura com uso de LIDMS é proposta no Capítulo 5.

Por fim, mas não menos importante, foi a especificação e implementação de um ambiente Web para execução de LIDMS. Este ambiente publica serviços Web REST e suporta a parametrização de planos de consulta que são executados em última instância pelo componente QEF-LD. Detalhes sobre o ambiente de execução podem ser encontrados no Capítulo 5.

1.4 Organização da Dissertação

Esta dissertação possui oito capítulos. Não sendo mais necessário tratar deste capítulo introdutório, os demais são delineados a seguir.

O Capítulo 2 – Fundamentação Teórica – apresenta uma síntese dos assuntos mais relevantes que servem de fundamentação para o entendimento dos demais capítulos desta dissertação. Ele expõe os principais conceitos e ferramentas relacionados a integração de dados, Consultas Federadas, Mediadores, Linked Data, Consultas sobre Linked Data, além de tratar também sobre a linguagem e a álgebra SPARQL.

O Capítulo 3 – Trabalhos Relacionados – trata das principais ferramentas existentes para lidar com a execução de consultas federadas sobre Linked Data, destacando funcionalidades e desvantagens de cada uma delas.

O Capitulo 4 propõe uma arquitetura de Linked Data Mashups baseada no uso de mediadores com três níveis de ontologias. Também são apresentados nesse Capítulo, o processo para a geração dos planos de consulta federados e o mecanismo de execução desses planos.

O Capítulo 5 explica o que são Linked Data Mashup Services (LIDMS) e propõe uma arquitetura de Linked Data baseada no seu uso. Além disso, o capítulo ainda trata do ambiente Web de execução de LIMDS que foi especificado e implementado como contribuição deste trabalho.

O Capítulo 6 trata da execução de planos de consulta federados sobre a web de dados através do uso do módulo de execução (QEF-LD) implementado. Esse capítulo explica o funcionamento do processador de planos de consultas, as extensões realizadas nele para permitir o acesso a dados no modelo RDF, os operadores propostos e os algoritmos utilizados nesses operadores.

O Capítulo 7 – Experimentos e Resultados – explica os experimentos realizados para avaliar a viabilidade do QEF-LD em comparação com outras estratégias de execução de consultas federadas, e analisa os resultados obtidos.

Por fim, o Capítulo 8 – Conclusão – tece as considerações finais sobre o trabalho e apresenta possíveis trabalhos futuros para dar prosseguimento ao que obtivemos até aqui.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Introdução

A Web atual deixou de ser apenas um espaço global de documentos interligados e está se tornando também um enorme espaço global de dados vinculados constituído de bilhões de triplas RDF que cobrem os mais variados domínios, denominada Web de Dados. *Linked Data* define um conjunto de princípios que formam a base para a difusão e uso de dados na Web. Desde 2007 conjuntos de dados dos mais diversos domínios têm sido publicadas de acordo com estes princípios, gerando um volume crescente de dados e, consequentemente, uma demanda por seu consumo.

A Web de Dados fornece um novo cenário à integração de dados, mas também traz novos desafios. Neste capítulo trataremos desses assuntos, cujo entendimento é essencial para a compreensão do contexto em que esta dissertação está inserida, bem como da fundamentação necessária ao entendimento dos capítulos seguintes.

2.2 Integração de Dados

Integração de dados é o processo de permitir acesso transparente à múltiplos sistemas de informação heterogêneos e distribuídos (LANGEGGER, 2010).

O principal objetivo dos sistemas de integração de dados é permitir que usuários consultem simultaneamente múltiplas fontes de dados heterogêneas, distribuídas e autônomas por meio de uma única interface de consultas, mantendo transparentes os procedimentos de acesso, extração e integração dos dados. Assim o sistema de integração de dados deve tratar de forma transparente problemas de heterogeneidade (estrutural, conceitual e tecnológica), distribuição e autonomia das fontes durante a execução de consultas, que são descritos a seguir.

Distribuição. As fontes de dados estão dispersas geograficamente, sendo interligadas por meio de uma rede de computadores. Como consequência da distribuição, é necessário lidar com os problemas envolvidos nas redes, como replicação, fragmentação e custo da transmissão dos dados e a capacidade de processamento de cada servidor.

Autonomia. Refere-se ao nível de independência de operação de cada fonte de dados que participe de um sistema de integração, em que as fontes possuem controle total sobre os dados e, geralmente, não podem afetar suas funcionalidades e requerer modificações.

Heterogeneidade. Como os esquemas das fontes são desenvolvidos independentemente, eles possuem estruturas e terminologias diferentes (heterogeneidade estrutural e semântica), o que ocorre tanto com os esquemas que vêm de domínios diferentes, quanto com os modelados no mesmo domínio do mundo real, pelo fato de ser desenvolvidos por pessoas diferentes, em diferentes contextos. Para serem efetivos, os sistemas de integração de dados devem ser capazes de transformar dados de diferentes fontes para responder a consultas feitas sobre esse esquema.

2.3 Abordagens para integração de dados

Os sistemas de integração de dados seguem, normalmente, duas abordagens clássicas segundo (LENZERINI, 2002): virtual ou materializada.

A abordagem materializada extrai os dados de fontes distintas e materializa-os localmente em repositórios chamados *datawarehouses*. As consultas são realizadas sobre a base materializada e, dessa forma, apresentam melhor desempenho em relação à abordagem virtual. A desvantagem dessa abordagem é a necessidade de manter a base materializada sempre atualizada. Em alguns casos, é necessário a presença de um administrador para isso, caso contrário poderão ser geradas respostas com informações desatualizadas.

Na abordagem virtual, os dados são recuperados diretamente das fontes quando o sistema de integração precisa responder a uma consulta, ou seja, sistemas que seguem essa abordagem enviam consultas diretamente às fontes de dados e, após os resultados individuais obtidos, integram os dados e fornecem o resultado final ao usuário ou à aplicação que os requisitou. Os dados acessados por essa abordagem estão sempre atualizados, entretanto, sabe-se que os custos de processamento das consultas e de acesso às fontes são fatores consideráveis na escolha.

A escolha da abordagem a ser seguida está diretamente relacionada à arquitetura e às características da aplicação de integração.

Neste trabalho, as soluções propostas para execução de consultas federadas sobre Linked Data usam a abordagem virtual.

2.4 Uso de mediadores para integração de dados

Gio Wiederhold, um dos proponentes da abordagem de mediadores, definiu mediador como sendo um módulo de software que explora o conhecimento codificado sobre alguns conjuntos ou subconjuntos de dados para criar informação para aplicações em uma camada de mais alto nível (WIEDERHOLD, 1992).

Os mediadores são os responsáveis por disponibilizar o esquema de global (esquema de mediação). Os usuários desses sistemas submetem as consultas a um esquema de mediação, necessitando conhecer apenas a estrutura do esquema e a linguagem de consulta adotada.

Ao receber uma consulta sobre o esquema, o mediador a reescreve em subconsultas que serão executadas pelas diversas fontes de dados e integra os resultados obtidos. Como as fontes podem ser baseadas em tecnologias, formatos e modelos completamente distintos, os mediadores normalmente se utilizam de tradutores (*wrappers*) para o acesso e execução das consultas sobre as fontes de dados. Após obter os resultados locais, os mediadores consolidam o resultado e retornam ao usuário a resposta da consulta realizada sobre o esquema de mediação.

2.5 Mashups

A palavra *mashup* foi usada primeiramente na área de música para definir uma música criada a partir da edição e combinação de outras. No domínio da computação *mashup* é um tipo de aplicação Web que combina conteúdo de múltiplos serviços ou fontes de dados em um novo serviço ou fonte de dados (THOR; AUMUELLER; RAHM, 2007). Esse tipo de aplicação normalmente é utilizado para necessidades situacionais específicas. Uma vez que combinar dados na Web é um processo dinâmico, ou seja, ocorre de acordo com uma entrada específica do usuário, mashups suportam apenas um conjunto restrito e predefinido de consultas, visando atingir tempos de execução reduzidos. A interação com um mashup tipicamente ocorre através de uma interface gráfica na web ou de um Serviço Web.

O site *ProgrammableWeb*[1] catalogava mais de 6.400 *mashups* em janeiro de 2012. Um exemplo clássico de *mashup* é a aplicação *HousingMaps*[2] que a partir de restrições como cidade e faixa de preço feitas pelo usuário em um formulário Web, obtém dados de imóveis do serviço *CraigsList*[3] e exibe suas localizações em uma página Web usando a API do *Google Maps*[4]. Outro exemplo é a aplicação *EveryBlock's Chicago*[5] que obtém dados publicados pelo Departamento de Polícia de Chicago e exibe os locais dos diferentes tipos de crime também através do *Google Maps*.

A integração de dados em *mashups* é normalmente dinâmica com operações simples sobre os dados para alcançar curto tempo de execução. Integrações de dados mais complexas não podem ser realizadas por grande parte das ferramentas existentes para criação de *mashups*. Uma estratégia para viabilizar o tempo de execução de um *mashup* é dividir uma consulta complexa em uma sequência de consultas que exigirão sucessivas interações com o usuário. A cada interação, o usuário refina a consulta, detalhando cada vez mais o resultado.

Atualmente, a atividade de desenvolver mashups enfrenta alguns problemas recorrentes, devido à forma como o conteúdo dinâmico é especificado: uma vez que o conteúdo deve ser especificado como consultas sobre fontes distribuídas, é necessário conhecimento específico do usuário sobre uma linguagem de consulta e/ou programação, bem como das URIs das fontes de dados e do vocabulário dessas fontes. Essa dificuldade é reforçada pela falta de uma linguagem de alto nível para consultar fontes heterogêneas.

A construção de mashups envolve muitas vezes a manipulação de diferentes APIs Web para acessar diferentes fontes e serviços (LORENZO et al., 2009). Tais APIs não possuem um padrão e não associam dados das fontes usadas, deixando os dados de fontes distintas ainda isolados entre si e, consequentemente, dificultando a tarefa de integração dos dados.

Finalmente, há também um problema relacionado à semântica: devido aos diversos vocabulários das fontes em questão, é difícil encontrar relações entre dados residentes nestas fontes.

[1] http://www.programmableweb.com/
[2] http://www.housingmaps.com/
[3] http://www.craigslist.org/
[4] http://maps.google.com/
[5] http://chicago.everyblock.com/

Arquitetura de mashups

De acordo com (MAXIMILIEN; RANABAHU; GOMADAM, 2008) um mashup pode ser visto como uma arquitetura de três níveis, consistindo de (i) um nível de **dados** relacionado a operações de manipulação, mediação e integração de dados de fontes heterogêneas. Consiste de todas as possíveis manipulações de dados (conversão, filtro, combinação, etc.) para integrar diferentes fontes de dados. O nível de dados usa um modelo interno único de representação dos dados, como o modelo RDF, por exemplo. Fontes de dados cujos dados são representados nesse modelo podem ser acessadas diretamente. No entanto, *wrappers* são usados para converter dados de diversos modelos de dados específicos para o modelo interno usado no nível de dados; (ii) um nível de **processo** que realiza a coreografia de diferentes APIs ou serviços entre as aplicações envolvidas para criar um novo processo. Essa composição pode ser implementada em alguma linguagem de programação ou de workflow. O nível de processo ainda é responsável pela lógica de negócio do *mashup* e pelo acesso aos dados do nível de dados; e (iii) um nível de **apresentação** que diz respeito à interface do usuário e permite exibição de conteúdo proveniente do nível de processo e interação com o usuário. Essa interface pode ser uma página Web ou um Serviço Web. A Figura 2.1 ilustra a arquitetura geral de um *mashup* composta pelos níveis de dados, processo e apresentação.

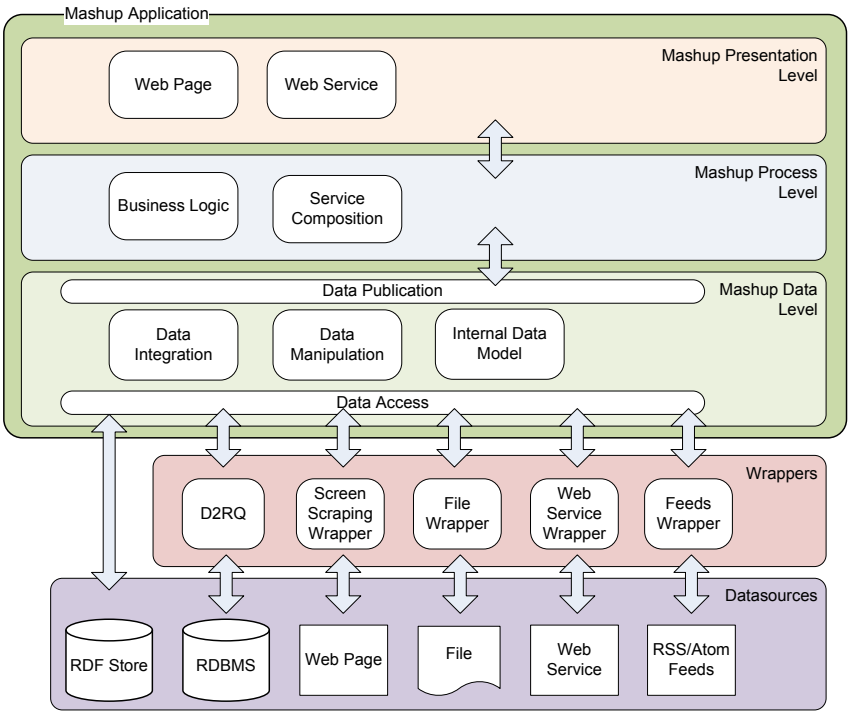

Figura 2.1: Arquitetura Geral de uma aplicação *mashup*

***Mashups* de Dados** Um *mashup* de dados pode ser especificado sobre o nível de dados da arquitetura citada, definindo: como os dados serão acessados; como os dados são definidos, de modo que possam ser descobertos e reusados posteriormente por outros mashups; como consultas são executadas para recuperação de dados; e como conjuntos de dados são acessados.

2.6 Linked Data

A Web atual deixou de ser apenas um espaço global de documentos interligados e está se tornando um enorme espaço global de dados vinculados constituído de bilhões de triplas RDF que cobrem os mais variados domínios (HEATH; BIZER, 2011). Esta nova Web, denominada Web de Dados, visa pavimentar o caminho para a Web Semântica funcional, onde haverá a disponibilidade de uma grande quantidade de dados vinculados em formato RDF. A Web Semântica fornece tecnologias para efetivamente publicar, recuperar e descrever dados distribuídos na Web. A integração de dados em grande escala é provavelmente uma das melhores aplicações para essas tecnologias. A Web de Dados baseia-se nos princípios *Linked Data* delineados pelo diretor geral do W3C, o pesquisador Tim Berners-Lee. De fato, *Linked Data* é um conjunto de melhores práticas para publicação e conexão de dados estruturados na Web que se baseia em tecnologias da Web Semântica, e que permite reduzir a complexidade de integração de dados devido às ligações estabelecidas e descritas entre os conjuntos de dados. Desse modo, Linked Data tem o potencial de facilitar o acesso aos dados semanticamente relacionados, estabelecendo conexões explícitas entre conjuntos de dados. Devido a esta conexão, aplicações Web que visam integrar dados, tais como Mashups, podem se beneficiar do uso de Linked Data.

Duas razões tornam Linked Data uma solução promissora para integrar dados semanticamente (RIBEIRO et al., 2011):

1. é mais fácil criar e manter um Linked Data Mashup do que um mashup convencional, uma vez que, no primeiro caso, as relações entre os vocabulários das fontes já são identificadas e uma linguagem de consulta de alto nível – como SPARQL – pode ser usada para consultar conjuntos de dados heterogêneos;

2. há uma quantidade crescente de dados RDF publicados na Web de acordo com princípios de Linked Data. No entanto, um problema recorrente é de como especificar automaticamente uma ordem para execução de uma consulta sobre diversos conjuntos de dados. Atualmente, embora existam soluções capazes de executar tais planos de consulta sobre conjuntos de dados de Linked Data, a tarefa de especificá-los ainda é executada manualmente pelo usuário. Portanto, há a necessidade de um processo comum, bem definido, que permita a construção dessa especificação, de forma simples.

A Web de Dados cria inúmeras oportunidades para a integração semântica de dados, fomentando o desenvolvimento de novos tipos de aplicações e ferramentas. Muito esforço tem sido despendido pela comunidade para o desenvolvimento de navegadores, mecanismos de busca e outras ferramentas específicas para consumo de dados vinculados. Além disso, muitas ferramentas para publicação de dados seguindo os princípios de *Linked Data* foram desenvolvidas e disponibilizadas, motivando a publicação de dados de forma aberta. Essas forças têm revelado desafios a ser superados para o uso efetivo da Web de Dados, o que tem aumentado o interesse de pesquisa nesta área.

2.6.1 Definição

Linked Data é um conjunto de melhores práticas para publicação e consumo de dados estruturados na Web, permitindo estabelecer ligações entre itens de diferentes conjuntos de dados para formar um único espaço de dados global (HEATH; BIZER, 2011). Os dados publicados na Web de acordo com essas melhores práticas podem ser processados por máquinas, possuem significado explicitamente definido e podem estar ligados a outras fontes de dados. (BIZER; HEATH; BERNERS-LEE, 2009) resume *Linked Data* como o uso da Web para criar ligações tipadas entre dados de diferentes conjuntos de dados.

2.6.2 Princípios e Boas Práticas

As melhores práticas relacionadas à *Linked Data* foram inicialmente propostas por (BERNERS-LEE, 2006) e ficaram conhecidas como os princípios de *Linked Data* que são enumerados a seguir:

1. Usar URIs como nomes para coisas.
2. Usar URIs HTTP para que as pessoas possam procurar esses nomes.
3. Quando alguém procurar uma URI, prover informação útil, usando os padrões (RDF, SPARQL).
4. Incluir links para outras URIs, de modo que possam permitir a descoberta de mais coisas.

Esses princípios fornecem a base para a publicação e interligação de dados estruturados na Web. Posteriormente, eles foram estendidos por documentos originados a partir das experiências da comunidade de *Linked Data* (BIZER; CYGANIAK; HEATH, 2007; SAUERMANN; CYGANIAK, 2008), resultando em boas práticas de publicação e consumo de *Linked Data*. Algumas dessas boas práticas relacionadas a *Linked Data* são tratadas na seção 2.6.7, após a exposição de alguns fundamentos necessários ao seu entendimento.

2.6.3 Dados Abertos

Tim Berners-Lee sugeriu uma forma de classificar o nível de abertura dos dados baseada em estrelas (BERNERS-LEE, 2006). Nesse tipo de classificação, os dados abertos podem ter entre uma e cinco estrelas. Assim, quanto mais abertos forem os dados, mais estrelas eles terão. Dados sob licença aberta possuem uma estrela. Se, além disso, os dados estiverem disponibilizados de forma estruturada, como em forma de planilha, por exemplo, eles terão duas estrelas. Três estrelas são atribuídas aos dados que usam formato não proprietário como CSV ou XML. Caso os dados estejam usando os padrões Web recomendados pelo W3C como modelo RDF, URIs e linguagem SPARQL, eles são classificados com quatro estrelas. Por fim, as cinco estrelas são atribuídas aos dados que além das características já mencionadas em relação as demais estrelas, ainda estão conectados com dados de outros conjuntos de dados através de links RDF.

2.6.4 Conjuntos de Dados

Um **conjunto de dados** é um conjunto de triplas RDF publicadas, mantidas ou agregadas por um único provedor. Para facilitar a identificação de conjuntos de dados relevantes na Web é importante que cada conjunto de dados publique metadados com informações de proveniência, que podem incluir o mantenedor do conjunto de dados, informações de licença e tópicos de assuntos relacionados ao conjunto de dados. O vocabulário padronizado pelo W3C para publicação de metadados sobre conjunto de dados disponíveis como Linked Data é o *VoiD – Vocabulary of Interlinked Datasets* (ALEXANDER et al., 2009, 2011). Além disso, os conjuntos de dados devem ser registrados em repositórios de informações sobre vários conjuntos de dados disponíveis, como por exemplo: CKAN[6], VOIDstore[7] e VoiD Browser[8]. A ferramenta WoDQA[9] – Web of Data Querying Application – pode ser usada para automatizar a identificação de conjuntos de dados nos repositórios de documentos VoiD a partir de padrões de triplas fornecidos.

2.6.5 Web de Documentos e Web de Dados

Para facilitar o entendimento da Web de dados, podemos estabelecer um paralelo com a Web de documentos que já conhecemos. A Web de dados pode ser acessada a partir de navegadores RDF, assim como os navegadores HTML são usados para acessar a Web de documentos. Enquanto na Web de documentos usamos links HTML para navegar entre diferentes páginas, na Web de dados os links RDF são usados para acessar dados de outras fontes. Portanto, os links de hipertexto são capazes de conectar os documentos, assim como os links RDF interligam os dados. A Tabela 2.1 ilustra algumas comparações entre a Web de Documentos e a Web de Dados.

Web de Documentos	**Web de Dados**
Navegadores HTML	Navegadores RDF
Links HTML conectando documentos	Links RDF interligando dados
Mecanismo de identificação – URIs	Mecanismo de identificação – URIs
Mecanismo de acesso – HTTP	Mecanismo de acesso – HTTP
Formato de conteúdo – HTML	Modelo de dados – RDF
—	Linguagem de consulta – SPARQL

Tabela 2.1: Comparações entre a Web de Documentos e a Web de Dados

Além disso, a Web de documentos está alicerçada em um pequeno conjunto de padrões: um mecanismo de identificação global e único (*URIs – Uniform Resource Identifiers*), um mecanismo de acesso universal (*HTTP – Hypertext Transfer Protocol*) e um formato de conteúdo amplamente usado (*HTML – Hypertext Markup Language*). De modo semelhante, a Web de dados também tem por base alguns padrões bem estabelecidos como: o mesmo mecanismo

[6] http://ckan.org/
[7] http://void.rkbexplorer.com/
[8] http://kwijibo.talis.com/voiD/
[9] http://seagentdev.ege.edu.tr:8180/seagent/wodqa/wodqa.seam

de identificação usado na Web de documentos (URIs), um modelo de dados comum (RDF) e uma linguagem de consulta para acesso aos dados (SPARQL). O modelo RDF (MANOLA; MILLER, 2004) é baseado na ideia de identificar os recursos da Web usando identificadores chamados *Uniform Resource Identifiers – URIs*[10], e descrever tais recursos em termos de propriedades, as quais podem apontar para outras URIs ou ser representadas por literais. Esses padrões serão abordados a seguir.

2.6.6 Padrões

Os padrões abertos adotados em *Linked Data* são amplamente difundidos e suportados pelas mais variadas linguagens de programação.

URI – Uniform Resource Identifier
Um Identificador Uniforme de Recursos (*URI – Uniform Resource Identifier*) é uma sequência compacta de caracteres que identifica um recurso físico ou abstrato (BERNERS-LEE; FIELDING; MASINTER, 2005). URIs são usadas no contexto de *Linked Data* para identificar objetos e conceitos, permitindo que eles sejam dereferenciados para obtenção de informações a seu respeito. Assim, uma URI dereferenciada resulta em uma descrição RDF do recurso identificado. Por exemplo, a URI *http://www.w3.org/People/Berners-Lee/card#i* identifica o pesquisador Tim Bernes-Lee.

Protocolo *HTTP – Hypertext Transfer Protocol*
O protocolo de Transferência de Hipertexto (*Hypertext Transfer Protocol*) é o mecanismo padrão de acesso aos documentos e dados na Web. É um protocolo genérico, sem estado e no nível de aplicação para sistemas distribuídos, colaborativos e hipermídia. Uma característica do HTTP é a tipagem e negociação de representação de dados, que permitem a construção de sistemas de forma independente dos dados transferidos (FIELDING et al., 1999).

RDF – Resource Description Framework
A utilização um modelo de dados comum – modelo RDF – torna possível a implementação de aplicações genéricas capazes de operar sobre o espaço de dados global (HEATH; BIZER, 2011). O modelo RDF (MANOLA; MILLER, 2004) é um modelo de dados descentralizado, baseado em grafo e extensível, possuindo um alto nível de expressividade e permitindo a interligação entre dados de diferentes conjuntos de dados. Ele foi projetado para a representação integrada de informações originárias de múltiplas fontes. Os dados são descritos na forma de triplas com sujeito, predicado e objeto, onde o sujeito é uma URI, o objeto pode ser uma URI ou um literal e o predicado é uma URI que define como sujeito e predicado estão relacionados. Por exemplo, a afirmação em português *'http://www.w3.org/People/Berners-Lee/card#i tem uma propriedade denominada creator cujo valor é Tim Bernes-Lee'* pode ser definida através de uma tripla RDF conforme ilustrado na Figura 2.2.

Cada tripla faz parte da Web de Dados e pode ser usada como ponto de partida para explorar esse espaço de dados. Triplas de diferentes conjuntos de dados podem ser facilmente combinadas para formar um único grafo. Além disso, é possível usar termos de diferentes

[10]http://www.ietf.org/rfc/rfc2396.txt

```
Sujeito: <http://www.w3.org/People/Berners-Lee/card#i>
Predicado: <http://purl.org/dc/elements/1.1/creator>
Objeto: "Tim Bernes-Lee"
```

Figura 2.2: Exemplo de tripla RDF

vocabulários para representar os dados. O modelo RDF ainda permite a representação de dados em diferentes níveis de estruturação, sendo possível representar desde dados semiestruturados a dados altamente estruturados.

Links RDF

No contexto de *Linked Data* os *Links RDF* descrevem relacionamentos entre dois recursos (HEATH; BIZER, 2011). Um *link RDF* consiste em uma tripla com três URIs. As URIs referentes ao sujeito e objeto identificam os recursos relacionados. A URI referente ao predicado define o tipo de relacionamento entre os recursos. Uma distinção útil que pode ser feita é com relação a *links* internos e externos. *Links RDF* internos conectam recursos dentro de um único conjunto de dados. *Links* externos conectam recursos servidos por diferentes conjuntos de dados *Linked Data*. No caso de *links* externos, as URIs referentes ao sujeito e predicado do link pertencem a espaço de nomes *(namespaces)* distintos. *Links* externos são cruciais para a Web dos Dados visto que eles permitem interligar as fontes de dados dispersas em um espaço global de dados.

A figura 2.3 apresenta dois exemplos de *links RDF*. O primeiro exemplo interliga o perfil FOAF do pesquisador Tim Berners-Lee localizado em um arquivo RDF ao recurso que o identifica na fonte de dados do DBLP. No segundo exemplo, o recurso que identifica Tim Berners-Lee na fonte DBpedia também é ligado ao recurso na fonte DBLP que o identifica. A propriedade *http://www.w3.org/2002/07/owl#sameAs* define que os recursos interligados representam a mesma entidade do mundo real.

```
Sujeito:   <http://www.w3.org/People/Berners-Lee/card#i>
Predicado: <http://www.w3.org/2002/07/owl\#sameAs>
Objeto:    <http://www4.wiwiss.fu-berlin.de/dblp/resource/person/100007>
```

```
Sujeito:   <http://dbpedia.org/resource/Tim\_Berners-Lee>
Predicado: <http://www.w3.org/2002/07/owl\#sameAs>
Objeto:    <http://www4.wiwiss.fu-berlin.de/dblp/resource/person/100007>
```

Figura 2.3: Exemplos de links RDF

O armazenamento de dados no modelo RDF pode ser realizado através de grafo em memória, arquivo texto ou banco de dados específico para armazenamento de triplas RDF, chamado de *RDF Store, Triple Store* ou *Quad Store*. Normalmente uma *Triple Store* é de fato uma *Quad Store*, pois suporta Grafos Nomeados *(Named Graphs)*. Um grafo nomeado é simplesmente uma coleção de triplas RDF nomeada por uma URI que identifica o grafo, com a finalidade de caracterizar a proveniência dos dados RDF. O armazenamento de triplas em

arquivo texto usa algum formato de serialização de RDF, como RDF/XML, Notation3 (N3), Turtle, NTriples ou RDF/JSON.

Protocolo e Linguagem SPARQL
Consultas à Web de Dados podem ser realizadas através da linguagem SPARQL (PRUD'HOMMEAUX; SEABORNE, 2008), que é a linguagem de consulta padrão da Web Semântica para recuperação de informações contidas em grafos RDF. No entanto, SPARQL não é somente uma linguagem de consulta declarativa, mas também um protocolo (CLARK; FEIGENBAUM; TORRES, 2008b) usado para enviar consultas e recuperar resultados através do protocolo HTTP. O uso da linguagem de consulta de alto nível SPARQL abstrai detalhes da sequência de passos necessária para a execução de consultas sobre conjuntos de dados heterogêneos.

Fontes *Linked Data* tipicamente fornecem um *SPARQL endpoint* que é um serviço Web com suporte ao protocolo SPARQL. Esse serviço possui uma URI específica para receber requisições HTTP com consultas SPARQL e retornar os resultados dessas consultas. Os resultados podem ter diferentes formatos. Consultas que usam os comandos SELECT e ASK geralmente são retornadas nos formatos XML, JSON ou texto plano. Já os resultados de consultas através dos comandos DESCRIBE ou CONSTRUCT normalmente usam os formatos RDF/XML, NTriples, Turtle ou N3. A maioria dos *endpoints SPARQL* exibem uma página HTML interativa que permite ao usuário digitar e submeter uma consulta.

Prefixos e Atalhos
Prefixos são bastante usados em consultas SPARQL e em alguns formatos de serialização de RDF para abreviar URIs. Assim se '`dc:`' é um prefixo para `<http://purl.org/dc/elements/1.1/>`, então '`dc:creator`' é uma notação abreviada da URI `<http://purl.org/dc/elements/1.1/creator>`. Um atalho frequentemente usado é a letra '`a`' ('um' em inglês) que serve para abreviar a URI '`rdf:type`', onde '`rdf:`' é um prefixo para `<http://www.w3.org/1999/02/22-rdf-syntax-ns#>`. A sintaxe '<>' é usada para identificar o próprio documento onde ela está inserida. O símbolo ponto-e-vírgula (';') é usado para delimitar uma tripla e definir que o sujeito da próxima tripla não deverá ser explicitamente definido, pois será o mesmo sujeito da tripla definida antes do símbolo ';'. A Figura 2.4 mostra trechos de grafos RDF serializados em formato Turtle. Os trechos são equivalentes, mas diferem na sintaxe, já que o primeiro trecho usa prefixos e atalhos, mas o segundo trecho não os utiliza.

A Web de dados ainda possui as seguintes características de acordo com (BIZER; HEATH; BERNERS-LEE, 2009):

1. É genérica e pode conter qualquer tipo de dado;
2. qualquer pessoa pode publicar dados;
3. não há restrições para seleção de vocabulários;
4. os dados são autodescritos, de modo que ao dereferenciá-los é possível obter sua definição;

Usando prefixos e atalhos:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix dc:   <http://purl.org/dc/elements/1.1/> .
@prefix :     <http://lia.ufc.br/~regispires/researchers.rdf#> .

<>  a                   foaf:Document ;
    dc:title            "Researchers file" .

:ufc
    a                   foaf:Organization ;
    foaf:name           "Universidade Federal do Ceará" .
```

Sem prefixos e atalhos:

```
<> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Document> .
<> <http://purl.org/dc/elements/1.1/title> "Researchers file" .

<http://lia.ufc.br/~regispires/researchers.rdf#ufc> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/0.1/Organization> .

<http://lia.ufc.br/~regispires/researchers.rdf#ufc> <http://xmlns.com/foaf/0.1/name>
"Universidade Federal do Ceará" .
```

Figura 2.4: Exemplos de grafos RDF equivalentes usando sintaxes distintas

5. o uso de um mecanismo padrão de acesso aos dados (HTTP) e de um modelo de dados padrão (RDF) simplifica o acesso aos dados;

6. as aplicações que usam a Web de dados não ficam limitadas a um conjunto fixo de fontes de dados, podendo inclusive descobrir novas fontes de dados em tempo de execução ao seguir links RDF.

2.6.7 Boas Práticas

A adoção de boas práticas relacionadas a *Linked Data* facilita a descoberta de informações relevantes para a integração de dados entre diferentes fontes. A seguir serão descritas algumas dessas práticas.

- **Selecionar URIs adequadas.** Devem-se evitar URIs contendo algum detalhe de implementação ou do ambiente em que estão publicadas. Como exemplo a evitar, consideremos o URI *http://lia.ufc.br:8080/ regispires/cgi-bin/resource.php?id=ufc* que possui detalhes da porta 8080 usada em seu ambiente de publicação e do script implementado em PHP necessário à sua execução.

  É frequente o uso de três URIs relacionadas a cada recurso: (i) um identificador para o recurso; (ii) um identificador para informações sobre o recurso para visualização através de navegadores HTML; (iii) um identificador para informações sobre o recurso em formato RDF/XML. A figura 2.5 representa um exemplo de três URIs relacionadas à pesquisadora Vânia Vidal na fonte DBLP. A representação de um recurso através diferentes URIs permite que a interface Linked Data realize o dereferenciamento da URI de acordo com o tipo de conteudo requisitado no cabeçalho HTTP (i.e. Text/HTML, application/rdf+xml, etc.).

```
http://dblp.l3s.de/d2r/resource/V%C3%A2nia_Maria_Ponte_Vidal
http://dblp.l3s.de/d2r/page/V%C3%A2nia_Maria_Ponte_Vidal
http://dblp.l3s.de/d2r/data/V%C3%A2nia_Maria_Ponte_Vidal
```

Figura 2.5: Exemplos de URIs relacionadas a um mesmo recurso

A figura 2.6 apresenta dois exemplos de requisições HTTP referente à URI da pesquisadora Vânia Vidal na fonte DBLP. No exemplo referente ao item (a), a requisição define como tipo MIME, dados no modelo RDF e recebe como resposta, através do redirecionamento 303, a URI referente aos dados da pesquisadora. No exemplo referente ao item (b), a requisição solicita os dados no formato HTML e recebe como resposta o redirecionamento para a URI referente à pagina HTML da pesquisadora.

```
(a)
$ curl -H "Accept: application/rdf+xml"
   http://dblp.l3s.de/d2r/resource/V%C3%A2nia_Maria_Ponte_Vidal

303 See Other: For a description of this item,
see http://dblp.l3s.de/d2r/data/V%C3%A2nia_Maria_Ponte_Vidal
```

```
(b)
$ curl -H "Accept: text/html"
   http://dblp.l3s.de/d2r/resource/V%C3%A2nia_Maria_Ponte_Vidal

303 See Other: For a description of this item,
see http://dblp.l3s.de/d2r/page/V%C3%A2nia_Maria_Ponte_Vidal
```

Figura 2.6: Exemplos de requisições HTTP com tipos MIME RDF e HMTL

- **Usar URIs dereferenciáveis** para que a descrição do recurso possa ser obtida da Web.
- **Utilizar URIs estáveis.** A alteração de URIs quebra links já estabelecidos, criando um problema para a localização de recursos. Para evitar esse tipo de alteração, recomenda-se um planejamento meticuloso das URIs que serão usadas e também que o responsável pela publicação detenha a propriedade do espaço de nomes.
- **Criar links para outras fontes de dados** de modo a permitir a navegação entre as fontes de dados. Os *links* podem ser criados de forma manual ou automatizada.
- **Publicação de Metadados.** Análise dos metadados facilita a seleção dos dados relevantes. Devem ser fornecidos metadados sobre proveniência e licenciamento dos dados. Também é recomendável a disponibilização de metadados sobre a fonte de dados. O vocabulário mais usado atualmente para publicação de metadados sobre conjunto de dados disponíveis é o *VoiD – Vocabulary of Interlinked Datasets*, conforme foi mencionado na Seção 2.6.4.
- **Utilizar termos de vocabulários amplamente usados**. Embora não haja restrições para seleção de vocabulários, é considerada uma boa prática o reuso de termos de vocabulários RDF amplamente usados para facilitar o processamento de *Linked Data* pelas aplicações

clientes (BIZER; CYGANIAK; HEATH, 2007). Novos termos só devem ser definidos se não forem encontrados em vocabulários já existentes. A seguir apresentamos alguns vocabulários bastante difundidos: *Friend-of-a-Friend* (FOAF), *Semantically-Interlinked Online Communities* (SIOC), *Simple Knowledge Organization System* (SKOS), *Description of a Project* (DOAP), *Creative Commons* (CC) e *Dublin Core* (DC). Uma relação mais extensa desses vocabulários é mantida pelo projeto *Linking Open Data* no *ESW Wiki*[11].

- **Estabelecer relações entre os termos de vocabulários proprietários para termos de outros vocabulários.** Isso pode ser feito através do uso das propriedades *owl:equivalentClass*, *owl:equivalentProperty*, *rdfs:subClassOf*, *rdfs:subPropertyOf*. A figura 2.7 mostra que a classe Pessoa de um vocabulário local é equivalente à definição da classe *Person* no vocabulário da *DBpedia*. A definição de relações entre vocabulários facilita a integração de dados que utilizam esses vocabulários.

```
<http://lia.ufc.br/Pessoa> owl:equivalentClass <http://dbpedia.org/ontology/Person> .
```

Figura 2.7: Relação de equivalência entre termo proprietário e termo da DBpedia

- **Explicitar formas de acesso adicional aos dados** como *endpoints SPARQL* e *RDF dumps*.

2.6.8 O Projeto Linking Open Data e outras iniciativas para fomentar a Web de Dados

Inúmeras iniciativas voltadas para fomentar a criação da Web de Dados surgiram nos últimos anos, como por exemplo, o projeto *Linking Open Data (LOD)*[12] que é um esforço comunitário iniciado em 2007 e suportado pelo W3C para identificar fontes de dados publicadas sob licenças abertas, convertê-las para RDF e publicá-las na Web usando os princípios de *Linked Data*. Até agosto de 2010, este projeto havia publicado 295 conjuntos de dados compostos por mais de 31 bilhões de triplas RDF e 500 milhões de *links* RDF englobando os mais variados domínios como informações geográficas, censo, pessoas, empresas, comunidades online, publicações científicas, filmes, músicas, livros, além de outros (BIZER; JENTZSCH; CYGANIAK, 2011). A figura 2.8 mostra um diagrama de nuvem[13] com as fontes de dados publicadas pelo projeto LOD e as interligações entre elas em setembro de 2011. O tamanho dos círculos corresponde ao número de triplas de cada fonte de dados. As setas indicam a existência de pelo menos 50 links entre duas fontes. A origem de uma seta indica a fonte que possui o link e a fonte referenciada é a fonte para a qual a seta está apontando. Setas bidirecionais representam fontes que se referenciam mutuamente. A espessura da seta corresponde ao número de links.

Outra importante iniciativa foi a criação do *Workshop Linked Data on the Web* (LDOW)[14] dentro da programação da *International World Wide Web Conference* (WWW2008),

[11] http://www.w3.org/wiki/TaskForces/CommunityProjects/LinkingOpenData/CommonVocabularies
[12] http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
[13] http://lod-cloud.net/
[14] http://events.linkeddata.org/ldow2008/

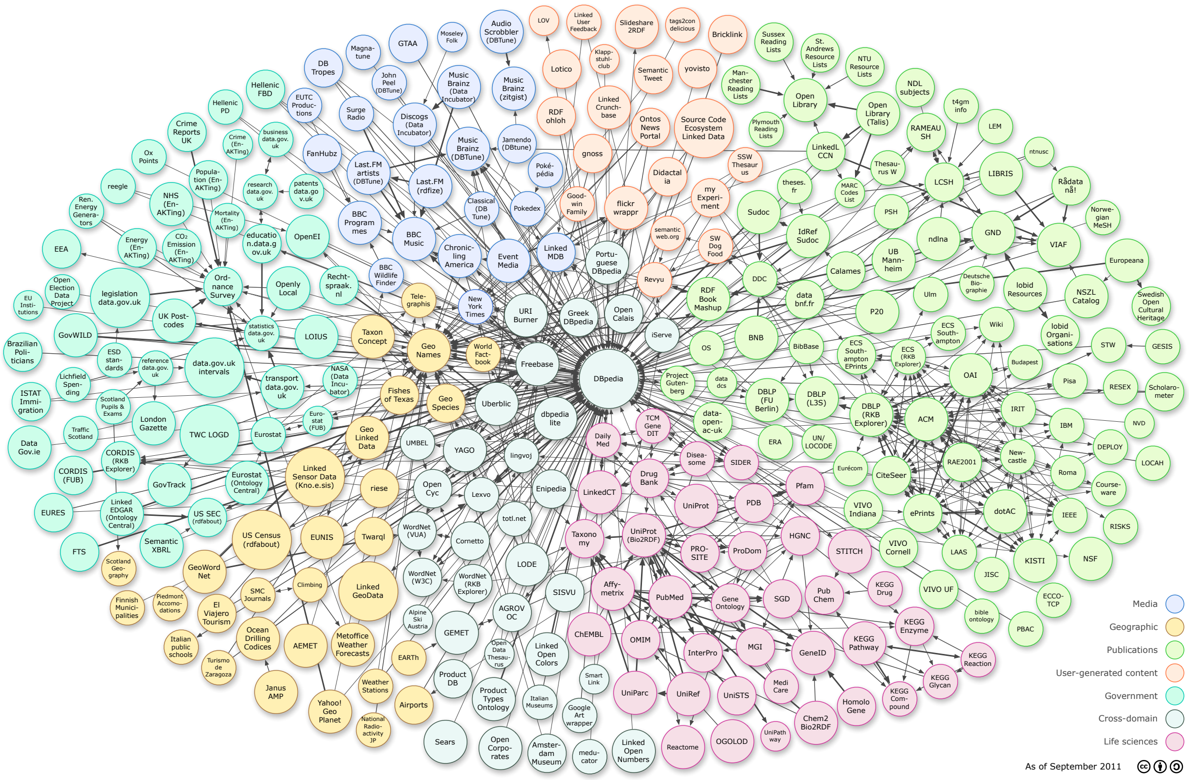

Figura 2.8: Diagrama de nuvem *Linking Open Data*, por Richard Cyganiak e Anja Jentzsch. Atualizado em 19/09/2011.

tendo, desde então, entre os membros de seu comitê organizador o pesquisador Tim Berners-Lee (W3C/MIT). No Brasil foi estabelecida no âmbito governamental a Infraestrutura Nacional de Dados Abertos (INDA)[15], uma importante iniciativa criada em 2011 para aplicar os princípios de *Linked Data* na publicação de dados governamentais abertos.

2.6.9 Conversão de dados para o modelo RDF e Wrappers RDF

Como RDF é o modelo de dados comum usado em *Linked Data*, os dados publicados devem estar nesse modelo ou serem convertidos para ele. Caso os dados não adotem o modelo RDF, há duas abordagens possíveis para tratar essa heterogeneidade:

(i) **Materialização dos dados no modelo RDF** – Usar um processo de **conversão**, onde os dados não RDF são usados para gerar um arquivo RDF através de uma ferramenta específica[16]. Desse modo, através de conversores específicos é possível converter bancos de dados relacionais, planilhas, arquivos CSV, arquivos XML e outros documentos para o formato RDF. O projeto *RDFizer* [17] contém informações de ferramentas para conversão de vários formatos de dados para RDF, além de hospedar algumas dessas ferramentas. Após geração do arquivo em formato RDF, seus dados podem ser carregados em uma *RDF Store* e publicados. Uma van-

[15]http://wiki.gtinda.ibge.gov.br/
[16]http://www.w3.org/wiki/ConverterToRdf
[17]http://simile.mit.edu/RDFizers/

tagem dessa abordagem é a melhoria de desempenho que pode ser obtida ao usar formas de armazenamento especificamente otimizadas para realizar a persistência de triplas RDF. No entanto, o armazenamento das triplas requer espaço extra em relação aos dados originais. Além disso, a conversão demanda certo tempo para ser realizada e os dados em RDF podem ficar desatualizados em relação aos dados originais.

(ii) **Fornecimento de uma visão RDF (abordagem virtual)** de dados que não estão no modelo RDF através de um *Wrapper RDF*. A conversão dinâmica realizada pelo *wrapper* baseia-se em mapeamentos estabelecidos entre o modelo nativo e o modelo RDF, devendo haver um *wrapper* específico para cada tipo de modelo. Um *Wrapper RDF* também pode prover uma visão RDF a dados que precisam ser acessados através de uma Web API. *RDF Book Mashup* (BIZER; CYGANIAK; GAUS, 2007) é uma aplicação *mashup* escrita em PHP que funciona como um *RDF Wrapper* usado para combinar dados obtidos a partir das APIs proprietárias *Amazon Web Service* e *Google Base*. Desse modo, informações sobre livros, autores, revisões e comparações de ofertas entre diferentes livrarias podem ser usados por clientes genéricos, incluindo navegadores e mecanismos de busca RDF. Essa abordagem tende a ter um desempenho inferior à abordagem anterior (i) devido às traduções dinâmicas entre os modelos que deve ser realizada a cada uso da visão RDF e também devido ao modelo original não estar otimizado especificamente para uso de triplas. No entanto, o uso de *Wrappers RDF* traz grandes vantagens, pois como o acesso ocorre sobre os dados originais, a visão RDF não requer espaço de armazenamento extra e não corre o risco de ter dados desatualizados.

A ampla difusão dos Bancos de Dados Relacionais motiva a necessidade de publicação dos dados no modelo relacional como *Linked Data*. Assim, seguindo a abordagem (ii), há *Wrappers RDB para RDF* que criam visões RDF a partir de mapeamentos entre as estruturas relacionais e os grafos RDF. A plataforma D2RQ[18] (BIZER; SEABORNE, 2004) fornece a infraestrutura necessária para acessar bancos de dados relacionais como grafos RDF virtuais. Ela disponibiliza uma interface *Linked Data*, *SPARQL endpoint* e *dumps RDF* baseados em um mapeamento declarativo entre o esquema do banco de dados e os termos RDF. A plataforma D2RQ possui os seguintes componentes:

- **Linguagem de mapeamento D2RQ** é uma linguagem declarativa para descrever as correspondências entre o modelo relacional e o modelo RDF. Os mapeamentos escritos em D2RQ são documentos RDF.
- **Mecanismo D2RQ** é um plug-in para os *frameworks Jena* e *Sesame* que usa os mapeamentos escritos na linguagem D2RQ para converter chamadas às APIs desses *frameworks* em consultas SQL ao banco de dados para obtenção dos resultados.
- **Servidor D2R**[19] (BIZER; CYGANIAK, 2006) é um servidor HTTP que usa o mecanismo D2RQ para prover uma interface *Linked Data* e um *SPARQL endpoint* sobre o banco de dados relacional.

[18] http://www4.wiwiss.fu-berlin.de/bizer/d2rq/spec/
[19] http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/

Os componentes anteriormente descritos funcionam de forma integrada como pode ser observado na figura 2.9 que apresenta a arquitetura da plataforma D2RQ.

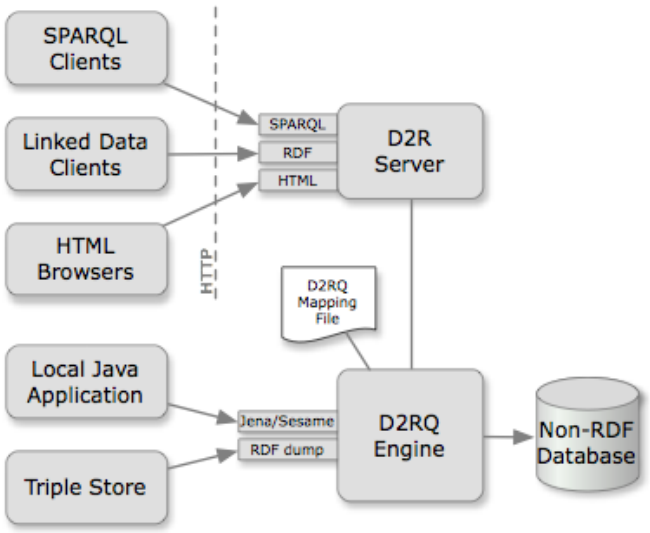

Figura 2.9: Arquitetura da plataforma D2RQ extraída de http://www4.wiwiss.fu-berlin.de/bizer/d2rq/spec/

Além do D2R, duas outras ferramentas se destacam como *Wrappers RDB para RDF*: o *Virtuoso RDF Views* (ERLING; MIKHAILOV, 2006) e o *Triplify*[20] (AUER et al., 2009). Este último é um pequeno *plugin* para aplicações Web que permite mapear os resultados de consultas SQL em RDF, JSON e *Linked Data*. Depois disso, os dados podem ser compartilhados e acessados na web de dados. *Triplify* consiste de poucos arquivos, totalizando menos de 500 linhas de código. Importante ressaltar que D2R e Triplify além de fornecerem uma visão RDF aos dados relacionais, também permitem a geração de dumps RDF (materialização dos dados).

Um importante passo para a padronização desse tipo de solução para lidar com o modelo relacional foi a criação em 2009 do grupo de trabalho RDB2RDF[21] do W3C. Desde então, o grupo tem definido a linguagem padrão R2RML (DAS; SUNDARA; CYGANIAK, 2011) visando o mapeamento de dados e esquemas relacionais para RDF, que tende a substituir as soluções de mapeamento já existentes.

2.6.10 Aplicações Linked Data

URIs, palavras-chave e consultas SPARQL são usados como ponto de partida para o consumo de *Linked Data*. Assim, todas as aplicações que consomem a Web de dados usam direta ou indiretamente pelo menos um desses itens.

Segundo (HEATH; BIZER, 2011) o consumo de *Linked Data* é realizado basicamente através de dois tipos de aplicações: **aplicações genéricas** que fazem uso de *Linked Data* de qualquer domínio e **aplicações de domínio específico** que são especificamente desenvolvidas para lidar com *Linked Data* relacionado a um determinado domínio.

[20] http://triplify.org/
[21] http://www.w3.org/2001/sw/rdb2rdf/

2.6.10.1 Aplicações Genéricas

A utilização de padrões Web e um modelo de dados comum torna possível a implementação de aplicações genéricas capazes de operar sobre o espaço de dados global. Essas aplicações permitem o consumo de dados relacionados a múltiplos domínios distribuídos pela web de dados. Ao percorrer os *links RDF* é possível explorar e descobrir novas informações. A seguir serão abordados alguns tipos de aplicações genéricas normalmente usadas para acessar *Linked Data*.

Navegadores *Linked Data*
Navegadores *Linked Data* são aplicações executadas a partir dos navegadores Web convencionais que dereferenciam URIs e exibem uma visualização desse resultado, possibilitando a navegação aos dados de fontes relacionadas a partir dos *RDF links*. O *LOD Browser Switch*[22] é uma aplicação web que obtém detalhes a respeito de uma URI especificada pelo usuário a partir da seleção de um dos vários navegadores *Linked Data* disponibilizados pela aplicação. Assim é possível comparar a visualização de uma URI através de vários desses navegadores.

Alguns dos navegadores *Linked Data* são citados a seguir: *Explorator*[23] (ARAÚJO; SCHWABE, 2009; ARAÚJO; SCHWABE; BARBOSA, 2009), *Disco Hiperdata Browser*[24], *Marbles* [25], *Tabulator*[26] (BERNERS-LEE et al., 2006, 2007), *LinkSailor*[27] e *Graphite RDF Browser*[28]. A figura 2.10 exibe informações obtidas a partir do dereferenciamento da URI *http://dblp.l3s.de/d2r/resource/authors/Marco_A._Casanova* pelo *Disco*.

Mecanismos de Busca Linked Data O acesso à Web de Dados pode ocorrer a partir de mecanismos de busca específicos capazes de realizar pesquisas que levam em consideração a semântica dos dados. Esses mecanismos de busca permitem localizar recursos de diferentes fontes normalmente através de palavras-chave. A consulta pode ser realizada pelo usuário através de uma interface web ou através de serviços web providos pelos mecanismos de busca. Mecanismos de busca *Linked Data* percorrem a Web de dados percorrendo os *links* entre as fontes de dados e fornecendo a possibilidade de consultas sobre os dados dessas fontes. Os resultados das buscas são URIs que podem ser dereferenciadas e visualizadas através dos navegadores RDF. Atualmente há vários mecanismos de busca *Linked Data*. A seguir citamos alguns deles: *Sindice*[29] (OREN et al., 2008), *VisiNav*[30], *Watson*[31] (D'AQUIN et al., 2007) e *Swoogle*[32] (DING et al., 2004).

Linked Data Mashups

[22] http://browse.semanticweb.org/
[23] http://www.tecweb.inf.puc-rio.br/explorator
[24] http://www4.wiwiss.fu-berlin.de/rdf_browser/
[25] http://www5.wiwiss.fu-berlin.de/marbles/
[26] http://dig.csail.mit.edu/2005/ajar/ajaw/tab
[27] http://linksailor.com/
[28] http://graphite.ecs.soton.ac.uk/browser/
[29] http://sindice.com/
[30] http://visinav.deri.org/
[31] http://watson.kmi.open.ac.uk/WatsonWUI/
[32] http://swoogle.umbc.edu/

Figura 2.10: Visualização de informações sobre recurso através do navegador Disco

Linked Data Mashups permitem aos usuários executar consultas e integrar dados estruturados e vinculados na web. Dados manipulados por *Linked Data Mashups*, em geral são dinamicamente recuperados através de um conjunto de especificações dos dados sobre fontes distribuídas e das definições de relacionamento entre estas fontes.

Especificar, construir e manter *Linked Data Mashups* não são tarefas fáceis, devido à necessidade de conhecer as URIs das fontes e o vocabulário utilizado por cada fonte, além da dificuldade para encontrar e estabelecer relações entre dados destas fontes, quando não estão vinculadas. Um dos objetivos deste trabalho é reduzir a complexidade da construção e manutenção de *Linked Data Mashups*.

Sig.ma[33] (TUMMARELLO et al., 2010) é um exemplo de *Linked Data Mashup* de uso genérico. Ele permite a busca de dados estruturados a partir de uma palavra-chave e os exibe em uma única página, integrando os dados de múltiplas fontes. A visão criada pelo *Sig.ma* baseia-se em resultados fornecidos pelo mecanismo de busca *Sindice*[34] (OREN et al., 2008). O usuário pode aprovar, rejeitar ou acrescentar fontes para estabelecer uma visão dos dados relevantes. Ao selecionar uma entidade da lista de resultados, uma nova visão é apresentada

[33] http://sig.ma/
[34] http://sindice.com/

ao usuário. Um *link* permanente pode ser criado para futuros acessos ou compartilhamento dessa visão. As filtragens das fontes de dados realizadas pelos usuários coletivamente ajudam a classificar melhor a relevância das fontes e aperfeiçoar a qualidade dos resultados futuros. Além da interface web do usuário, *Sig.ma* ainda provê uma API destinada aos desenvolvedores de aplicações. A figura 2.11 ilustra o resultado de uma consulta sobre a pesquisadora Vânia Vidal envolvendo dezesseis fontes, onde quatro delas foram rejeitadas.

Figura 2.11: Visão criada pelo Sig.ma sobre a pesquisadora Vânia Vidal

Outras aplicações genéricas

Informações adicionais sobre determinado recurso podem ser obtidas através da localização de objetos referenciados pelas propriedades `rdfs:seeAlso` e `owl:sameAs`. Serviços online de coreferenciamento como o *sameAs*[35] são usados para encontrar URIs de diferentes fontes de dados que representam um mesmo conceito.

LDSpider[36] é um framework capaz de navegar pela web de dados seguindo *links* para obter dados de fontes *Linked Data* e os armazenar em uma *RDF Store* através de SPARQL Update ou como arquivo RDF.

2.6.10.2 Aplicações de domínio específico

Várias aplicações têm sido desenvolvidas para integrar *Linked Data* em domínios específicos. Essas aplicações também podem ser classificadas como *Linked Data Mashups*, mas voltados para um determinado domínio. A seguir descreveremos algumas delas.

[35] http://sameas.org
[36] http://code.google.com/p/ldspider/

Revyu[37] é uma aplicação web para crítica e classificação de qualquer item passível de avaliação. *Revyu* também disponibiliza uma API e um *SPARQL endpoint* para serem usados pelos desenvolvedores de aplicações.

DBpedia Mobile[38] (BECKER; BIZER, 2008) é uma aplicação cliente para dispositivos móveis consistindo de uma visão com um mapa e do navegador *Linked Data Marbles*. Baseado na localização geográfica de um dispositivo móvel, a aplicação exibe um mapa indicando localizações próximas a partir de dados extraídos das fontes *DBpedia*, *Revyu* e *Flickr*. O acesso ao *Flickr* é realizado através de um *Wrapper*. O usuário pode explorar informações sobre essas localizações e navegar em conjuntos de dados interligados. Também é possível a publicação de informações como *Linked Data*, de modo que possam ser usadas por outras aplicações.

Talis Aspire[39] é uma aplicação web voltada para que alunos e professores e possam encontrar os principais recursos educacionais em universidades do Reino Unido. O serviço é gratuito e provê ferramentas para criar e editar listas de leitura, além da produção e publicação de materiais educativos. Quando o usuário publica conteúdo, a aplicação cria triplas RDF em uma *RDF store*. Itens publicados são interligados de forma transparente a itens correspondentes de outras instituições.

BBC Programmes[40] e *BBC Music*[41] são projetos desenvolvidos pela *BBC Audio and Music Interactive*. A aplicação web *BBC Programmes* disponibiliza informações detalhadas sobre tipos, séries e episódios de todos os programas de TV e rádio transmitidos pela BBC. *BBC Music* fornece informações sobre artistas, vinculando-os aos programas da BBC. Assim é possível escolher um artista e obter todos os episódios de programas relacionados a ele. As aplicações mencionadas usam *Linked Data* como tecnologia de integração de dados, inclusive fazendo uso de vocabulários amplamente conhecidos como *DBpedia* e *MusicBrainz*.

Uso de Visões de Integração

O uso de visões de integração reduz a complexidade do acesso a múltiplos conjuntos de dados heterogêneos. Isso ocorre devido ao encapsulamento dos detalhes relacionados aos conjuntos de dados integrados pela visão. A seguir são apresentados dois enfoques para disponibilização dessas visões de integração.

Enfoque Materializado

No enfoque materializado a informação integrada é carregada em um repositório central. No contexto de Linked Data isso significa que triplas correspondentes à visão de integração são geradas, armazenadas e publicadas a partir dos mapeamentos entre os conjuntos de dados usados e a visão de integração.

Enfoque Virtual

No enfoque virtual, acessos à visão de integração são convertidos dinamicamente e de forma

[37] http://revyu.com/
[38] http://beckr.org/DBpediaMobile/
[39] http://www.talisaspire.com/
[40] http://www.bbc.co.uk/programmes
[41] http://www.bbc.co.uk/music

transparente em acessos aos conjuntos de dados integrados. Assim a integração ocorre no momento em que a consulta é executada (*on-the-fly integration*). Arquiteturas com ou sem o uso de mediador podem ser usadas nesse enfoque. A arquitetura com o uso de mediador permite a execução de consultas *ad-hoc* que são executadas pelo mediador. Na arquitetura sem o uso de mediador as consultas devem ser predefinidas e um plano de execução previamente criado é executado para integrar os dados. As arquiteturas com e sem o uso de mediador serão explicadas em detalhes nos Capítulos 4 e 5, respectivamente.

2.6.11 APIs para manipulação de Linked Data

A seguir descreveremos algumas APIs para manipulação de dados na web semântica que são usadas no desenvolvimento de aplicações de domínio genérico ou específico para consumo de *Linked Data*.

Sesame[42] e *Jena*[43] são *frameworks* de web semântica implementados em Java que fornecem APIs para manipulação de grafos RDF. Ambos possuem processador de consultas com suporte a consultas federadas em SPARQL (PRUD'HOMMEAUX; BUIL-ARANDA, 2011). A maioria dos mediadores e demais ferramentas que lidam com federação de consultas SPARQL usam internamente uma dessas duas APIs.

Sesame permite armazenamento, consulta e manipulação de dados RDF. Além disso, o *framework* é extensível e configurável em relação a formas de armazenamento (memória e *RDF store*), mecanismos de inferência, formatos de arquivo RDF e linguagens de consulta (SPARQL e SeRQL).

Jena foi desenvolvido no *HP Labs* entre 2000 e 2009. Atualmente faz parte do projeto *Apache* e suas principais características são: suporte a RDF, RDFa, RDFS, OWL e SPARQL; armazenamento de triplas RDF em memória, banco de dados relacional *(Jena SDB)* ou *RDF store (Jena TDB)*; processamento de consultas SPARQL *(Jena ARQ)*; disponibilização de *SPARQL endpoint* (*Joseki* ou *Fuseki*); disponibilização de mecanismos de inferência embutidos e interfaces para mecanismos de inferência externos.

Named Graphs API for Jena (NG4J)[44] é uma extensão ao framework Jena para análise, manipulação e serialização de conjuntos de grafos nomeados representando os grafos como modelos ou grafos do Jena. NG4J permite o armazenamento de grafos em memória ou em banco de dados. Consultas SPARQL podem ser realizadas sobre os grafos nomeados.

O *Semantic Web Client Library* (SWClLib)[45] (HARTIG; BIZER; FREYTAG, 2009) faz parte do NG4J e é capaz de representar a web de dados como um único grafo RDF. Ele recupera informações dereferenciando URIs, seguindo *links rdfs:seeAlso* e consultando o mecanismo de busca *Sindice*. O SWClLib considera todos os dados como um único conjunto global de grafos nomeados, sendo usado na implementação de vários navegadores *Linked Data*. Os

[42] http://www.openrdf.org/
[43] http://jena.apache.org/
[44] http://www4.wiwiss.fu-berlin.de/bizer/ng4j/
[45] http://www4.wiwiss.fu-berlin.de/bizer/ng4j/semwebclient/

grafos recuperados são mantidos em um *cache* local para melhorar o desempenho de buscas futuras.

ARQ2[46] é uma biblioteca escrita em PHP que contempla armazenamento de Triplas RDF, *SPARQL endpoint* e interface *Linked Data* em uma única ferramenta. As triplas RDF são armazenadas em um banco de dados MySQL. A infraestrutura necessária para o funcionamento do ARQ2 é muito simples por requerer apenas um servidor Web com suporte a PHP e um banco de dados MySQL, sendo facilmente encontrada em qualquer serviço de hospedagem Web.

2.6.12 Abordagens para execução de consultas sobre múltiplas fontes de dados

Aplicações podem acessar *Linked Data* na web através de consultas a um *SPARQL endpoint* de um determinado conjunto de dados. Embora esse acesso possa prover dados valiosos para a aplicação, essa abordagem ignora o grande potencial da web de dados, pois não explora as possibilidades deste imenso espaço de dados que integra um grande número de conjuntos de dados interligados. Essas possibilidades podem ser alcançadas pela execução de consultas complexas e estruturadas sobre múltiplos conjuntos de dados. (HARTIG; LANGEGGER, 2010) discutem diferentes abordagens para realizar essas consultas sobre a web de dados, classificando-as basicamente em dois tipos: tradicionais e inovadoras.

Abordagens Tradicionais
Data warehousing e federação de consultas são abordagens amplamente discutidas na literatura de banco de dados para realização de consultas sobre dados distribuídos em fontes autônomas. Consultas sobre a web de dados podem utilizar essas abordagens tradicionais que requerem o conhecimento prévio das fontes de dados relevantes e, portanto, limitam as fontes de dados que serão levadas em conta para obter as respostas de uma consulta. A seguir descreveremos a aplicação dessas abordagens sobre a web de dados.

Data warehousing usa uma base de dados centralizada que coleta e armazena os dados das fontes. No contexto de *Linked Data*, podem-se materializar dados das fontes relevantes em uma base centralizada para a execução de consultas sobre ela. Tal estratégia também pode ser usada em mecanismos de busca sobre a web de dados. Além disso, ela possui o melhor desempenho dentre as abordagens que serão aqui discutidas já que os dados podem ser acessados diretamente na base centralizada, sem a necessidade de comunicações adicionais através da rede. No entanto, em fontes de dados cujo volume de dados é muito grande, a materialização dos dados tende a requerer bastante tempo e espaço de armazenamento. Outro problema é que atualizações sobre as fontes não são imediatamente refletidas sobre o repositório central, podendo ocasionar consultas com resultados desatualizados em relação aos dados originais. Outra questão a ser considerada é que as consultas somente são realizadas sobre os dados materializados e não sobre toda a web de dados.

Federação de consultas baseia-se na distribuição do processamento de consultas para múltiplas fontes de dados autônomas. Seu objetivo é dar ao usuário acesso aos dados por meio de algum vocabulário padrão especificado em uma ontologia de domínio. Consultas po-

[46] http://arc.semsol.org/

dem ser formuladas baseadas nessa ontologia. Um mediador decompõe, de forma transparente, a consulta em subconsultas, direciona as subconsultas a múltiplos serviços de consulta distribuídos, e, finalmente, integra os resultados das subconsultas. Em mais detalhe, o processamento de uma consulta requer as seguintes tarefas: particionamento, adaptação, mapeamento, otimização e execução. A tarefa de adaptação consiste na modificação e extensão da consulta, por exemplo, através da inclusão de termos similares ou mais abrangentes, a partir de relacionamentos com outros vocabulários, expandindo assim o escopo do espaço de busca de forma a obter melhores resultados. A tarefa de mapeamento consiste na seleção conjuntos de *Linked Data* que têm potencial para retornar resultados para as expressões contidas na consulta. A tarefa de otimização avalia o custo de diferentes estratégias para processar a consulta, preparando um plano de execução para a consulta. Finalmente, a tarefa de execução implementa uma via de comunicação com os conjuntos de *Linked Data* e processa o plano de execução preparado pela tarefa de otimização, possivelmente adaptando-o dinamicamente. Uma vantagem da federação de consultas é que ela não requer tempo ou espaço adicional para materialização de dados. Por outro lado, a execução de consultas é mais lenta devido às transmissões de rede necessárias para realização das subconsultas sobre as fontes de dados. Além disso, as consultas não podem ser realizadas sobre toda a web de dados, mas somente sobre as fontes de dados registradas no mediador. *DARQ* (QUILITZ; LESER, 2008) é um mediador baseado no processador de consultas *Jena ARQ* capaz de realizar consultas distribuídas sobre a web dados. *SemWIQ* (LANGEGGER, 2010) é outro mediador que estende o *Jena ARQ* a fim de consultar a web de dados fazendo uso de estatísticas (LANGEGGER; WÖSS, 2009) para otimizar as consultas. FedX (SCHWARTE et al., 2011a, 2011b) é um mediador que estende o Framework Sesame com uma camada de federação que possibilita o processamento eficiente de consultas sobre fontes distribuídas de Linked Data. (VIDAL et al., 2011) apresentam um *framework* baseado em mediador de três níveis para integração de dados sobre *Linked Data*. Desafios relacionados à eficiência de consultas federadas e uma abordagem para otimização dessas consultas baseada em programação dinâmica foram tratados por (GÖRLITZ; STAAB, 2011).

Abordagens Inovadoras

As abordagens inovadoras surgiram para eliminar a restrição imposta pelas abordagens tradicionais de limitarem as consultas sobre as fontes previamente conhecidas. Assim, elas permitem a descoberta das fontes durante a execução das consultas, podendo atuar sobre toda a web de dados. (HARTIG; LANGEGGER, 2010) caracterizam duas abordagens inovadoras: descoberta ativa baseada em federação de consultas e consultas exploratórias (também conhecidas como *link traversal*).

Descoberta ativa baseada em federação de consultas é uma estratégia baseada na combinação de processamento de consultas federado com uma descoberta ativa de fontes de dados relevantes pode ser usada para possibilitar o uso de fontes de dados desconhecidas. Essa estratégia parece não ter sido implementada até o momento da publicação do presente trabalho, mas é uma estratégia que vale a pena ser objeto de investigações futuras, desde que pode combinar as vantagens da federação de consultas com a possibilidade de obter dados de fontes ainda desconhecidas pelo mediador.

Consultas exploratórias *(link traversal)*. No enfoque exploratório, proposto por

(HARTIG; BIZER; FREYTAG, 2009) dados são descobertos e recuperados em tempo de execução da consulta. Este enfoque é baseado na busca de URIs, onde uma consulta SPARQL é executada através de um processo interativo onde URIs são dereferenciadas de modo a recuperar suas descrições em RDF na Web e os resultados da consulta construídos a partir dos dados recuperados. Desse modo, consultas exploratórias seguem *RDF links* para obter mais informações sobre os dados já existentes. Através do uso de dados recuperados a partir das URIs usadas em uma consulta como ponto de partida, o processador de consultas avalia partes da consulta. Soluções intermediárias resultantes dessa avaliação parcial geralmente contêm URIs adicionais que possuem ligações para outros dados que por sua vez, podem prover novas soluções intermediárias para a consulta. Para determinar o resultado completo da consulta, o processador de consultas avalia as partes da consulta e dereferencia URIs. O conjunto de dados usado na consulta é continuamente ampliado com dados potencialmente relevantes da web, cuja descoberta é realizada a partir das URIs de soluções intermediárias que podem estar em espaços de nomes distintos.

2.6.13 Desafios para integração de dados sobre Linked Data

A integração de dados de um grande número de fontes de dados heterogêneos na web ainda é complexa e ocorre no nível de vocabulário e identidade (links sameAs). No nível vocabulário a integração deve ocorrer entre diversos vocabulários distintos. No nível de identidade é possível ter identificadores e conceitos distintos interligados através de links 'owl:sameAs' para um mesmo conceito do mundo real.

(HARTIG; LANGEGGER, 2010) afirmam a necessidade de tornar mais transparente a integração de dados entre múltiplas fontes. Isso requer mapeamentos entre termos de diferentes vocabulários usados por fontes de dados com conteúdos similares. Além disso, pode ser necessário aplicar técnicas de fusão de dados para obter uma representação consistente de dados descritos diferentemente em fontes distintas, bem como, ajudar a resolver questões relacionadas a conflitos e qualidade dos dados. Muito ainda precisa ser feito também em relação à inferência e descoberta de conhecimento em dados provenientes de múltiplas fontes.

Permitir o mapeamento dos diversos vocabulários existentes, para que seja possível identificar e escolher dados de fontes diferentes sobre uma mesma entidade também é uma questão que requer maior aprofundamento.

Criação, edição e manutenção de *Linked Data* por vários usuários são desafios. Outro desafio está relacionado à manutenção desses dados para evitar problemas de acesso a informações que não estejam mais disponíveis. A Web de Dados é dinâmica e deve permitir que aplicações possam fazer atualizações e utilizar técnicas avançadas para a detecção de inconsistências. A web de dados é alimentada com dados provenientes dos mais diversos domínios, causando problemas quanto à confiabilidade e qualidade daquilo que é disponibilizado.

As possibilidades criadas por esses dados integrados podem infringir os direitos de privacidade dos usuários. Proteger os direitos dos indivíduos se torna difícil, pois os dados estão em fontes descentralizadas e sob diversas jurisdições legais. Prover ferramentas para explicitar

os direitos de cópia e reprodução sobre os dados é uma das lacunas no contexto de *Linked Data*.

Já existem várias aplicações funcionais e em desenvolvimento que permitem consultas complexas na Web de Dados, porém, ainda existem muitas oportunidades de pesquisa relacionadas à forma que os usuários poderão navegar por esses dados para tornar essa interação mais intuitiva, simples e objetiva.

Há algumas formas de consulta sobre múltiplas fontes *Linked Data*. Pode-se usar materialização dos dados em uma base centralizada, consultas federadas ou consulta exploratória *(link traversal)*. No entanto, ainda é necessário aperfeiçoar ou mesmo integrar esses tipos de acessos para tirar proveito das vantagens de cada um.

Determinar as informações mais relevantes, assim como detectar sua validade para melhorar a qualidade da informação, também são desafios que precisam ser superados através de algum *feedback* do usuário ou mesmo de forma automatizada.

Encontrar *endpoints SPARQL* relevantes normalmente é uma tarefa complexa devido à falta de descrição conceitual das fontes de dados. Para simplificar essa tarefa, é possível adotar as estratégias abordadas na Seção 2.6.4. No entanto, muito ainda precisa ser realizado para reduzir ainda mais a complexidade da descoberta dessas fontes.

2.7 Linguagem e Álgebra SPARQL

A especificação da linguagem SPARQL (PRUD'HOMMEAUX; SEABORNE, 2008) define um processo para execução de consultas SPARQL que consiste em uma sequência de passos, cujo ponto de partida é uma string representando a consulta SPARQL. Essa string é transformada em uma forma de sintaxe abstrata que é posteriormente convertida em um plano de consulta composto de operadores da álgebra SPARQL. O plano de consulta é, então, executado sobre um conjunto de dados RDF.

A seguir trataremos de alguns conceitos básicos necessários ao entendimento das operações da álgebra SPARQL.

O conjunto de **Termos RDF** usados na linguagem SPARQL é dado pela união de IRIs (*Internationalized Resource Identifiers*), Literais RDF e *blank nodes*. O conjunto de **IRIs** (DÜRST; SUIGNARD, 2005) é um subconjunto das URIs que omite espaços. *Literais RDF* podem ser planos (strings simples) ou tipados. Os **literais tipados** possuem um rótulo de tipo de dado e pode ser usado para representar números, datas, valores booleanos, etc. RDF usa os tipos de dados XML, mas também permite a definição de tipos personalizados. O literal tipado `"2004"^^<http://www.w3.org/2001/XMLSchema#int>` representa o número inteiro `2004`. Literais planos podem ter um rótulo para definir a língua usada. Desse modo, o literal `"livro"@pt` indica que o literal `"livro"` está escrito na língua portuguesa. ***Blank nodes*** são recursos anônimos que não possuem uma URI explícita. Ao serializar grafos RDF, *blank nodes* recebem um identificador único local gerado de forma aleatória.

Uma variável em SPARQL é uma variável de consulta (PRUD'HOMMEAUX; SEABORNE, 2008). Na sintaxe de SPARQL, o nome de uma variável é precedido pelo símbolo

'?'.

Um **conjunto de dados RDF** é um conjunto na forma:
$\{G, (< u_1 >, G_1), (< u_2 >, G_2), ..., (< u_n >, G_n)\}$, onde G e G_i são grafos e cada $< u_i >$ é um IRI distinto. G é o grafo padrão e $(< u_i >, G_i)$ são chamados de grafos nomeados.

Variáveis de consulta SPARQL possuem um ponto de interrogação (?) como prefixo e são usadas em padrões de triplas para estabelecer correspondências com o conjunto de dados RDF. Um **padrão de triplas** é um membro do conjunto:
$(T \cup V) \times (I \cup V) \times (T \cup V)$, onde T é o conjunto dos termos RDF, I é o conjunto dos IRIs e V é o conjunto das variáveis de consulta.

Um conjunto de padrões de triplas é chamado de **Padrão de Grafo Básico** (*Basic Graph Pattern – BGP*). BGPs são resolvidos como uma junção de padrões de triplas.

O **grafo ativo** é o grafo do conjunto de dados usado para realizar correspondência de padrão de grafo básico (*Basic Graph Pattern – BGP*).

O resultado de uma consulta é uma **sequência de solução** e é constituído das correspondências entre os dados e o padrão de grafo da consulta.

SPARQL possui quatro **formas de consulta** que são usadas para construir conjuntos de resultados (*result sets*) ou grafos RFD a partir das soluções encontradas. As formas de consulta são listadas a seguir:

- `SELECT` - Retorna todas ou um subconjunto das variáveis ligadas em uma correspondência de padrão de consulta.
- `CONSTRUCT` - Retorna um grafo RDF construído a partir da substituição de variáveis em um conjunto de *templates* de triplas.
- `ASK` - Retorna um valor booleano indicando se um padrão de consulta possui algum resultado ou não.
- `DESCRIBE` - Retorna um grafo RDF que descreve o recurso encontrado.

Os resultados dos modificadores `SELECT` e `ASK` frequentemente são serializados e retornados nos formatos JSON, XML, CSV ou TSV de acordo com os seguintes documentos: *Serializing SPARQL Query Results in JSON* (CLARK; FEIGENBAUM; TORRES, 2008a), *SPARQL Query Results XML Format* (BECKETT; BROEKSTRA, 2008) e *SPARQL 1.1 Query Results CSV and TSV Formats* (SEABORNE, 2011).

A especificação 1.0 da linguagem SPARQL define modificadores de sequência de solução que são usados para realizar algum tipo de alteração nos resultados da consulta. Os modificadores disponíveis são listados a seguir:

- **Order by**: ordena as soluções.
- **Projection**: seleciona determinadas variáveis.

- **Distinct**: não permite duplicidade de resultados, garantindo que cada resultado seja único no conjunto das soluções encontradas.
- **Reduced**: permite que soluções duplicadas possam ser eliminadas. No entanto, soluções duplicadas ainda poderão existir nos resultados. *Reduced* pode eliminar alguns, todos ou mesmo nenhum dos resultados duplicados. É útil na eliminação de resultados duplicados que são facilmente detectados, tornando-se uma operação computacionalmente menos cara que operação *Distinct*.
- **Offset**: define a posição da sequência a partir da qual as soluções serão consideradas.
- **Limit**: restringe o número de soluções consideradas.

A especificação 1.0 de SPARQL ainda define que string de consulta SPARQL é transformada em uma representação em forma de árvore que pode conter os seguintes elementos:

- **Padrões**: Termos RDF, Padrões de Triplas, Padrões de Grafos Básicos, Grupos, OPTIONAL, UNION, GRAPH e FILTER.
- **Modificadores**: DISTINCT, REDUCED, PROJECT, ORDER BY, LIMIT e OFFSET.
- **Formas de consulta**: SELECT, CONSTRUCT, DESCRIBE e ASK.

A extensão de consultas federadas (PRUD'HOMMEAUX; BUIL-ARANDA, 2011) à especificação SPARQL 1.1 (HARRIS; SEABORNE, 2012) define o elemento SERVICE que permite combinar dados distribuídos na Web. `SERVICE` define que um fragmento de consulta SPARQL seja executado sobre um *endpoint SPARQL* remoto.

`BINDINGS` é um recurso de SPARQL 1.1 que permite atribuir valores a variáveis SPARQL com a finalidade de restringir a quantidade de resultados de uma consulta.

`SERVICE` e `BINDINGS` são as únicas funcionalidades relacionadas à especificação SPARQL 1.1 referenciadas no contexto desta dissertação. Importante ressaltar que a maioria dos *endpoints SPARQL* disponíveis até o ano corrente (2012) não proveem suporte à especificação 1.1 de SPARQL.

Uma consulta SPARQL é uma tupla (E, C, F), onde E é uma expressão em álgebra SPARQL; C é um Conjunto de Dados RDF; e F é uma Forma de Consulta (`SELECT`, `CONSTRUCT`, `ASK` ou `DESCRIBE`).

A representação em árvore da consulta SPARQL é, então, convertida em uma consulta em álgebra SPARQL também representada através de uma árvore e que pode conter os seguintes símbolos da álgebra SPARQL:

- **Padrão de Grafo**: BGP (conjunto de padrões de triplas resolvido através de junções entre esses padrões de triplas), Join, LeftJoin, Filter, Union e Graph.

- **Modificadores de Solução**: ToList (usado para converter resultados de correspondência de padrão de grafo para sequências), OrderBy, Project, Distinct, Reduced e Slice (combinação de `OFFSET` e `LIMIT`).

Para cada símbolo da álgebra SPARQL é definido um operador que será usado em sua avaliação. Desse modo, os operadores servem para avaliar os nós da árvore que representa a consulta em álgebra SPARQL. Além dos símbolos anteriormente mencionados, o operador *Service* é referenciado nesta dissertação e foi adicionado à álgebra SPARQL de acordo com o documento que trata sobre extensão de consultas federadas (PRUD'HOMMEAUX; BUILARANDA, 2011).

2.8 Conclusões

Este capítulo apresentou uma síntese dos assuntos mais relevantes que servem de fundamentação para o entendimento dos demais capítulos desta dissertação. Foram expostos os principais conceitos e ferramentas relacionados a integração de dados, Consultas Federadas, Mediadores, Linked Data, Consultas sobre Linked Data, além de tratar também sobre a linguagem e a álgebra SPARQL.

3 TRABALHOS RELACIONADOS

3.1 Introdução

Esta seção apresenta algumas outras propostas existentes para execução de consultas federadas sobre Linked Data, com ou sem o uso de mediadores.

3.2 Estratégias para integração de dados sobre Linked Data sem uso de mediadores

Jena ARQ
O processador de consultas SPARQL *Jena ARQ*[1] é integrado ao framework de web semântica Jena[2]. Embora não seja uma solução abrangente de processamento distribuído de consultas, ele implementa suporte a execução de consultas federadas, conforme o documento que estabelece um padrão oficial para federação de consultas em SPARQL 1.1 (PRUD'HOMMEAUX; BUIL-ARANDA, 2011). De acordo com esse padrão, o operador SERVICE permite a determinação da URI do SPARQL endpoint, bem como da consulta SPARQL que será executada naquele endpoint. No entanto, a especificação é bastante simples e não prevê otimizações ou outras estratégias para melhoria do desempenho das consultas. Jena ARQ é uma das estratégias de execução de consultas federadas usada em nossos experimentos, conforme detalhado no Capítulo 7.

Sesame
O framework de web semântica Sesame[3], assim como o framework Jena, também possui um processador de consultas que implementa o padrão para federação de consultas definido no SPARQL 1.1 (PRUD'HOMMEAUX; BUIL-ARANDA, 2011). Ele também é usado como estratégia para execução de consultas federadas em nossos experimentos.

MashQL
(JARRAR; DIKAIAKOS, 2010) propõem uma linguagem de alto nível – *MashQL* – para permitir o desenvolvimento de mashups usando tecnologias da Web Semântica de forma visual, simples e intuitiva. *MashQL* adota RDF como modelo interno de dados e permite o desenvolvimento de *mashups* através de um navegador Web usando conexões entre componentes à semelhança do que é feito no *Yahoo Pipes*. Além disso, possibilita a formulação interativa de consultas sobre múltiplas fontes por usuários com conhecimentos limitados em TI. A linguagem é tão expressiva quanto SPARQL e as consultas são primeiramente traduzidas em consultas SPARQL antes de sua execução. No entanto, *MashQL* requer a tecnologia semântica do Oracle 11g para materializar os dados de todas as fontes usadas. Nessa estratégia há um grande custo de tempo e espaço para materialização dos dados, além de fornecer resultados desatualizados em relação às fontes. A sincronização dos dados materializados em relação às fontes também é outra questão não resolvida. MashQL também não possui licença livre, nem é disponibilizado

[1] http://jena.apache.org/documentation/query/
[2] http://jena.apache.org/
[3] http://www.openrdf.org/

publicamente para download, dificultado seu uso e aceitação por pessoas interessadas.

Semantic Web Pipes (SWP)
Semantic Web Pipes (SWP) (LE-PHUOC et al., 2009) apresenta um estilo de arquitetura flexível para o desenvolvimento de mashups de dados usando as tecnologias da Web Semântica. Pipes são planos de consultas criados visualmente pelo desenvolvedor através da conexão de operações sobre os dados no modelo RDF. Depois de criado, um Pipe é salvo e disponibilizado para uso através de uma simples requisição HTTP a uma URI específica usando Serviços Web REST.

Uma desvantagem de MashQL e Semantic Web Pipes é a construção manual do fluxo de operações do *mashup* pelo desenvolvedor. Além disso, as operações disponíveis para criação do fluxo de operações não seguem uma álgebra padrão, como a álgebra SPARQL.

Link Traversal
Link Traversal é um modelo proposto por (HARTIG; BIZER; FREYTAG, 2009) que explora a estrutura de navegação de *Linked Data*. A ideia principal é descobrir dados relevantes para responder a uma consulta durante a execução. A descoberta é feita percorrendo as ligações (RDF links) entre as fontes de dados RDF; e os resultados parciais são armazenados em um repositório local. Esta abordagem possui duas limitações: recupera apenas URIs e exige que a consulta seja executada a partir de uma URI somente, que faz o papel de padrão para a consulta. Finalmente, o fato do enfoque ser centralizado limita a otimização do processamento de consultas (REDDY; KUMAR, 2010). SQUIN[4] (HARTIG; BIZER; FREYTAG, 2009) é uma interface de consulta sobre *Linked Data* que implementa a abordagem de consultas exploratórias.

Networked Graphs
Networked Graphs (SCHENK; STAAB, 2008) segue uma abordagem baseada em visões de integração e fornece um formalismo que permite aos usuários definir grafos RDF usando visões sobre os outros grafos, possibilitando a composição de vários grafos que podem ser consultados de forma integrada. A composição é descrita de forma declarativa usando uma extensão da semântica do SPARQL. Uma desvantagem dessa solução é que seu funcionamento depende da extensão proposta à linguagem SPARQL que não está disponível nos endpoints SPARQL atualmente existentes, inviabilizando, assim, seu uso na web de dados.

3.3 Estratégias para integração de dados sobre Linked Data com uso de mediadores

DARQ – Distributed ARQ
Jena ARQ foi estendido pelo DARQ (QUILITZ; LESER, 2008). O principal objetivo é possibilitar a federação de consultas SPARQL, fornecendo acesso transparente a múltiplos SPARQL endpoints. Resumidamente, o DARQ decompõe uma consulta SPARQL em subconsultas e as submete aos SPARQL endpoints correspondentes, sendo os resultados obtidos de cada subconsulta integrados no mediador. Do ponto de vista arquitetural, DARQ é um mediador similar à arquitetura definida em (WIEDERHOLD, 1992). Sua arquitetura é representada na Figura 3.1. No entanto, diferente de um sistema baseado em mediador, o DARQ não prevê a integração de

[4] http://squin.sourceforge.net/

esquemas.

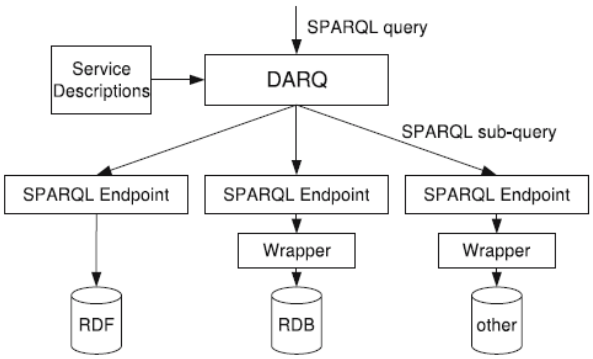

Figura 3.1: Arquitetura de integração de dados do mediador DARQ. Fonte: (QUILITZ; LESER, 2008)

Para usar o DARQ é necessário ter previamente armazenada a capacidade de cada fonte de dados em um arquivo chamado de descrição de serviço (service descriptions), um catálogo em RDF que descreve as fontes de dados. Essas capacidades são armazenadas localmente no mediador e descrevem os predicados presentes em cada fonte de dados com suas cardinalidades e também definem algumas informações de seletividades para expressões de filtro.

A reformulação da consulta é bem simples: a consulta principal é decomposta (particionada) em várias subconsultas de acordo com as informações obtidas na descrição de serviço. Cada uma delas deve ser respondida por um SPARQL endpoint conhecido.

Para otimização do processamento de consulta, o DARQ aplica otimização lógica e física. A otimização lógica usa regras para reescrever a consulta original antes da geração do plano; e a otimização física utiliza programação dinâmica interativa, empregando informações fornecidas pelo catálogo (service description) para definir a ordem de execução da junção.

DARQ não possui um serviços para registro e monitoramento de fontes de dados, dificultando o registro de novas fontes de dados, bem como a obtenção de estatísticas atualizadas sobre elas. A configuração do DARQ requer que o usuário explicitamente forneça uma arquivo de configuração contendo as descrições do serviço. Uma limitação do DARQ é que os predicados de todas as consultas necessariamente devem estar definidos, não sendo possível o uso de variáveis SPARQL como predicado nas consultas.

DARQ surgiu como uma prova de conceito no HP Labs em 2006. Seu autor diz que ele ainda está em estágio de desenvolvimento ainda muito inicial, não sendo indicado para uso em ambiente de produção. A última atualização do projeto ocorreu em 2008, e o autor afirma em seu blog pessoal [5] que não pretende continuar seu desenvolvimento.

SemWIQ – Semantic Web Integrator and Query Engine

O SemWIQ (LANGEGGER, 2010) é outro sistema de integração de dados em que as consultas são expressas em SPARQL. Também foi implementado usando o processador de consultas Jena ARQ. É baseado em uma arquitetura de mediadores-wrappers para geração do plano de

[5] http://blog.quilitz.de/2010/01/darq-federated-sparql-queries-status/

execução, adotando estratégia própria de otimização.

O sistema foi desenvolvido com foco no compartilhamento de dados científico e faz parte de um projeto maior chamado *Semantic Data Access Middleware for Grids (GSDAM)*, para permitir que dados científicos fornecidos por diferentes grupos de pesquisa sejam acessados de maneira transparente e de forma compartilhada. No entanto, SemWIQ pode servir genericamente como sistema de integração sobre Linked Data.

A arquitetura foi desenvolvida considerando três princípios básicos: os dados podem ser estruturados (por exemplo, usando XML, RDF/RDFS, OWL), são geograficamente distribuídos e armazenados em formatos heterogêneos.

O funcionamento do SemWIQ é explicado resumidamente a seguir. (i) O cliente estabelece conexão com o mediador e submete uma consulta SPARQL ao esquema de mediação; (ii) o tradutor calcula um plano de execução que é modificado pelo otimizador; (iii) o otimizador analisa a consulta e procura no catálogo as fontes de dados relevantes para responder à consulta. A saída do otimizador é um plano global para a execução da consulta que é enviado ao engine de execução; (iv) o engine delega a execução das subconsultas aos SPARQL endpoints; (v) quando a fonte de dados não tem suporte nativo a consultas SPARQL é preciso um wrapper capaz de reescrever a consulta original, escrita em SPARQL, para o formato específico da fonte de dados consultada; (vi) o mediador provê um wrapper local quando a fonte de dados é acessada por um serviço Web para envio dos dados; (vii) o catálogo armazena descrições e estatísticas sobre as bases de dados registradas. Essas estatísticas são geradas pela ferramenta RDFStats (LANGEGGER; WÖSS, 2009); (ix) finalmente, o componente de monitoramento atualiza as estatísticas sobre as fontes de dados registradas que enviam periodicamente consultas SPARQL aos endpoints a fim de gerar estatísticas.

A última atualização realizada no SemWIQ ocorreu em 2010 e há uma nota no site do projeto[6] afirmando que ele não é mais mantido.

FedX – Linked Data in a Federation

FedX (SCHWARTE et al., 2011a, 2011b) é um framework para acesso transparente a fontes de dados através de federação de consultas. Ele oferece processamento de consultas federadas eficiente e usa os padrões e protocolos suportados pela maioria dos endpoints SPARQL disponíveis atualmente. O FedX estende o framework Sesame com uma camada dedicada à federação. A infraestrutura do Sesame permite que fontes de dados heterogêneas sejam usadas como endpoints no contexto da integração de dados. A Figura 3.2 representa a arquitetura de aplicação que faz uso do FedX.

A camada de aplicação provê o frontend para o processador de consulta e é necessária para qualquer tipo de interação com a federação. A segunda camada é composta do Sesame e provê a infraestrutura básica para o processador de consultas, incluindo recursos de tradução de consultas, mapeamentos Java, componentes de entrada e saída, e a API para interação com o cliente. A camada de federação é implementada como uma extensão do Sesame e constitui o FedX, que adiciona as funcionalidades necessárias para gerenciamento de fontes de dados,

[6] http://semwiq.sourceforge.net/

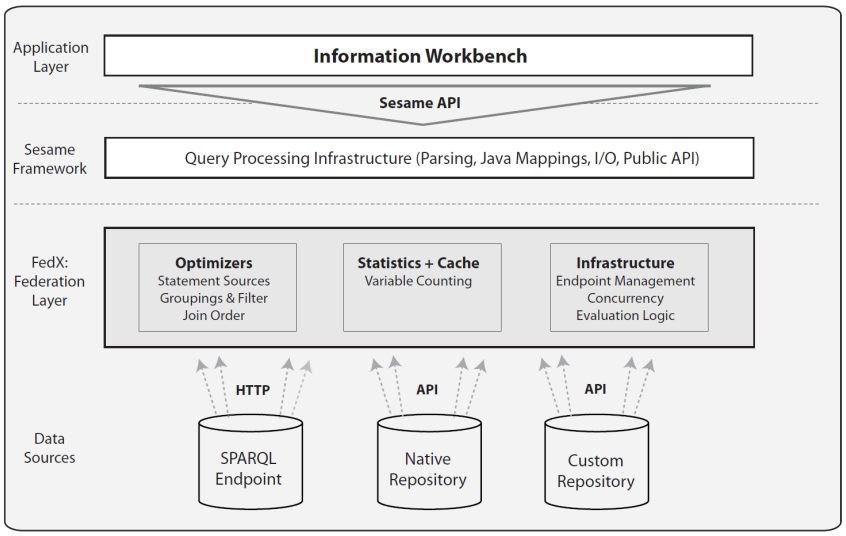

Figura 3.2: Arquitetura de uma aplicação usando FedX. Fonte: (SCHWARTE et al., 2011a)

comunicação com endpoints e, principalmente, otimizações para processamento de consultas distribuído. Fontes de dados podem ser adicionadas na forma de implementação de repositórios do Sesame. Implementações padrões são providas para repositórios local, nativo (Sesame) e endpoints SPARQL remotos. Com isso, é as seguintes federações são possíveis: puramente local consistindo de repositórios Sesame nativos, federações de endpoints ou formas híbridas.

O processamento de consultas no FedX consiste dos passos a seguir. Uma consulta global é formulada sobre a federação de fontes de dados. A consulta global é traduzida e otimizada para um plano de consulta federado, constituído de subconsultas que podem ser respondidas pelas fontes de dados individuais. Os resultados dessas consultas locais são integrados pelo federador e, finalmente, retornados. O processo é completamente transparente para o usuário, dando a impressão de que os dados estão virtualmente integrados em um único grafo RDF.

FedX vai além da federação de consultas definidas para SPARQL 1.1. Ele possibilita a configuração dinâmica de federações sobre fontes de dados distribuídas e executar consultas SPARQL padrões transparentemente sobre as diferentes fontes de dados, mesmo sem o uso das extensões de federação, como a operação SERVICE.

O uso de técnicas de otimização são cruciais para o bom desempenho do processador de consultas. Especialmente no cenário distribuído, é essencial aplicar abordagens para reduzir o número de chamadas aos endpoints remotos. A combinação de uma otimização própria para ordenação de junções e agrupamento de subconsultas, reduzem bastante o número de resultados intermediários e requisições, sendo as principais contribuições do FedX para a melhoria no desempenho de consultas federadas. O FedX implementa as seguintes técnicas de otimização:

- Fontes de sentenças RDF - examina fontes de sentenças RDF usando consultas ASK em SPARQL;

- Execução antecipada de filtros (o mais cedo possível)
- Processamento paralelo - uso de *threads* em operações de junção e união.
- Ordem de junções - reordenação de junções, técnicas de contagem de variáveis e heurísticas são usadas para estimar o custo de cada junção. Seguindo uma abordagem gulosa, as junções são executadas em ordem crescente de custo;
- Uso de *block nested loop joins* (similar ao algoritmo *SetBindJoin* usado neste trabalho)
- Agrupamento de sentenças RDF de uma mesma fonte de dados para execução em uma única consulta SPARQL a um endpoint.

Todas as funcionalidades do FedX são compatíveis com os endpoints compatíveis com SPARQL 1.0, sendo adequado aos ambientes atuais. O FedX não requer metadados pre-processados como estatísticas ou índices, o que o torna adequado para o processamento sob demanda de consultas *ad-hoc*.

3.4 Conclusões

Este capítulo apresentou as principais ferramentas existentes para lidar com a execução de consultas federadas sobre Linked Data, destacando funcionalidades e desvantagens de cada uma delas. Pretende-se com esse estudo, obter subsídios para a formulação de contribuições relevantes ao contexto de execução de consultas federadas. Essas contribuições serão apresentadas nos capítulos seguintes deste trabalho.

4 ARQUITETURA DE LINKED DATA MASHUPS BASEADA NO USO DE MEDIADORES

4.1 Introdução

A abordagem de mediadores permite aos usuários consultarem simultaneamente diversas fontes de dados distribuídas através de uma única interface de consultas, que provê transparência nos procedimentos de acesso, extração e combinação dos dados de cada fonte (WIEDERHOLD, 1992). Dessa forma, os usuários desses sistemas não submetem suas consultas aos esquemas das fontes de dados, mas a uma visão de mediação virtual, necessitando conhecer apenas o esquema da visão de mediação e a linguagem de consulta adotada. Ao receber uma consulta, o mediador a reescreve em subconsultas que serão executadas pelas diversas fontes de dados que estão sendo integradas.

Nesse trabalho adotamos o framework de mediação baseado em três níveis de ontologias proposto em (PINHEIRO, 2011). No framework proposto, o esquema mediado é representado por uma ontologia de domínio, a qual provê uma representação conceitual da aplicação. Cada fonte relevante é descrita por uma ontologia fonte, publicada na Web de acordo com os princípios de Linked Data. Cada ontologia fonte é reescrita em uma ontologia de aplicação, cujo vocabulário é um subconjunto do vocabulário da ontologia de domínio. Essa arquitetura de três níveis permite dividir a definição de mapeamentos em dois estágios: mapeamentos locais e mapeamentos de mediação, conforme mostrado na figura 4.1. Nessa arquitetura, o processo de reescrita de consulta é dividido em dois passos. Primeiro a consulta é decomposta usando os mapeamentos mediados, em um conjunto de subconsultas expressas em termos da ontologia de aplicação. Depois, as subconsultas são reescritas, usando os mapeamentos locais, em termos das ontologias fontes.

É importante frisar que, neste trabalho, podem ser utilizados os resultados de (SACRAMENTO et al., 2010), que definiram uma estratégia para a geração de ontologias de aplicação com os respectivos mapeamentos.

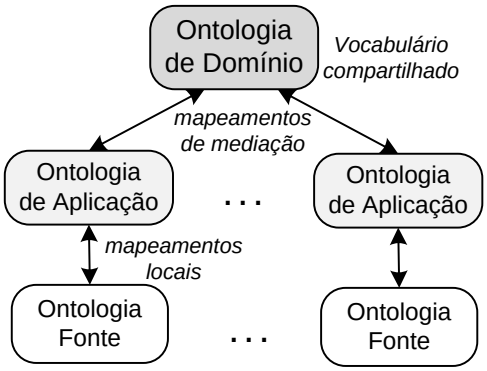

Figura 4.1: Arquitetura de três níveis baseada em ontologias. Adaptado de (SACRAMENTO et al., 2010)

4.2 Estudo de Caso

Para melhor entendimento dos conceitos expostos, apresentamos um estudo de caso de um mashup na área de saúde para integrar informações sobre drogas e doenças, a partir de algumas fontes de dados publicamente disponíveis na Web de acordo com os princípios de Linked Data (BERNERS-LEE, 2006). Estes dados são obtidos de cinco fontes de dados que estão disponíveis publicamente na Web.

Ontologia de Domínio

A Figura 4.2 mostra a representação conceitual da Ontologia de Domínio (OD) definida para a aplicação de mashup de Drogas e Doenças (D&D) com o prefixo `ddg:` usado em sua representação. A OD foi definida a partir da ontologia definida no projeto *Linking Open Drug Data (LODD)*[1] pelo grupo de interesse *Semantic Web for Health Care and Life Sciences* do W3C. Esse grupo verificou que existiam muitos dados publicamente disponíveis da área de saúde e resolveu publicá-los de acordo com os princípios de Linked Data conforme apresentado por (JENTZSCH et al., 2009).

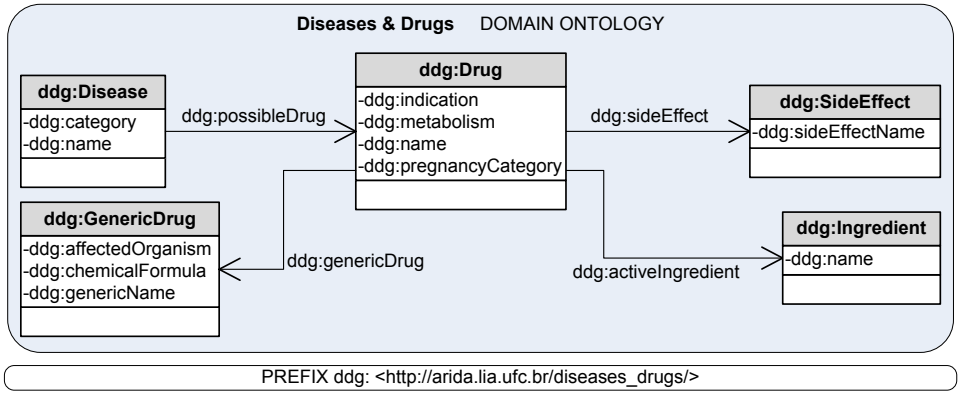

Figura 4.2: Ontologia de Domínio para o mashup D&D

Ontologias das Fontes e Mapeamentos Locais

O Mashup D&D integra dados de cinco fontes de dados que estão disponíveis publicamente na Web: *Diseasome, DailyMed, DrugBank, DBpedia* e *Sider*, conforme ilustrado na Figura 4.3. Cada uma das fontes tem associado um Endpoint SPARQL e seguem os princípios de Linked Data.

As fontes *Diseasome, DailyMed, DrugBank* e *Sider* publicam uma visão RDF sobre dados armazenados em banco de dados relacional através de um wrapper D2R. A visões RDF publicadas usam o mesmo vocabulário da ontologia de domínio do mashup D&D. Neste trabalho omitiremos detalhes das ontologias exportadas pelos endpoints das fontes, e dos mapeamentos utilizados pelo wrapper D2R para gerar as visões RDF. Informações mais detalhadas podem ser obtidas em (JENTZSCH et al., 2009).

Ontologias de Aplicação e Mapeamentos de Mediação

A Figura 4.4 mostra a representação conceitual das ontologias de aplicação (OAs) definidas

[1] http://www.w3.org/wiki/HCLSIG/LODD

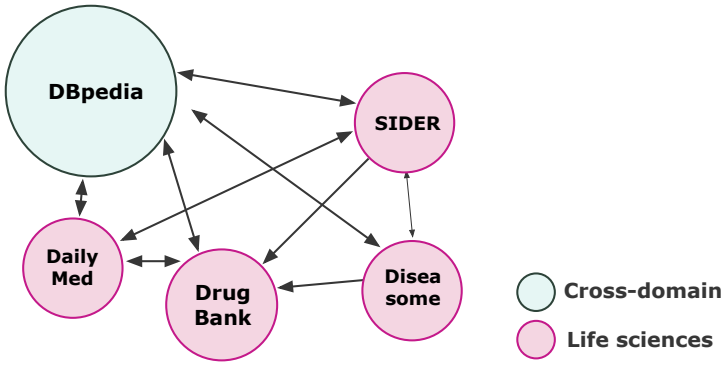

Figura 4.3: Fontes de dados usadas no Mashup D&D

a partir das Ontologias das fontes, conforme o processo proposto em (SACRAMENTO et al., 2010). Os prefixos usados nas OAs são apresentados na Figura 4.5. A classe *Drug* está presente em todas as OAs, exceto *diseasome*. As instâncias de Drug das OAs que representam uma mesma droga são interligadas através da propriedade `owl:sameAs` ou `dmed:genericDrug`.

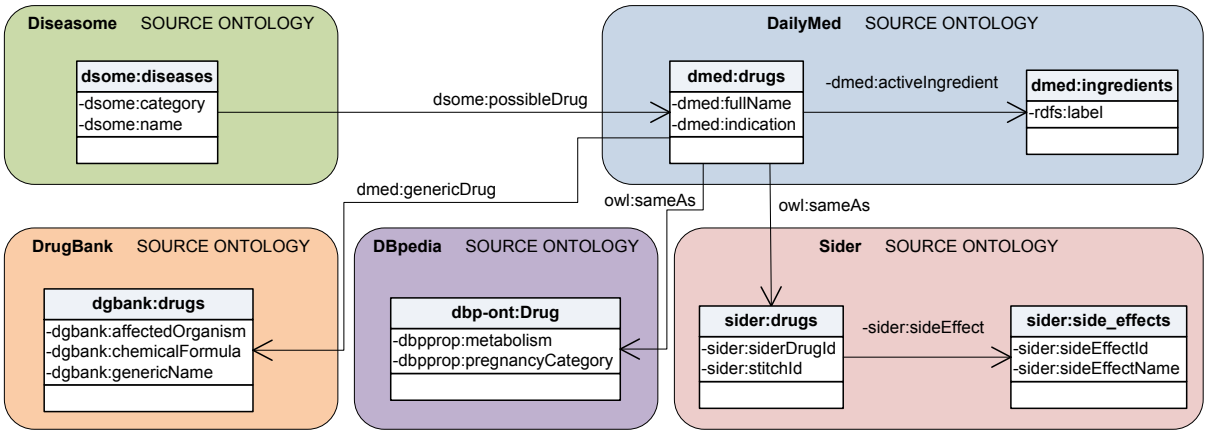

Figura 4.4: Ontologias de Aplicação do mashup D&D.

As regras de mapeamento das OAs para a OD são apresentadas na Figura 4.6. Os mapeamentos estão definidos através do formalismo de mapeamento baseado em regra apresentado em (VIDAL et al., 2011). Através desse formalismo é possível definir classes ou propriedades virtuais, que aparecem na cabeça de cada regra.

```
dbp-ont: <http://dbpedia.org/ontology/>
dbpprop: <http://dbpedia.org/property/>
dgbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
dmed: <http://www4.wiwiss.fu-berlin.de/dailymed/resource/dailymed/>
dsome: <http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/>
owl: <http://www.w3.org/2002/07/owl#>
rdfs: <http://www.w3.org/2000/01/rdf-schema#>
sider: <http://www4.wiwiss.fu-berlin.de/sider/resource/sider/>
```

Figura 4.5: Prefixos usados nas *OAs*.

ddg:Disease(D) ⇐ dsome:diseases(D) ddg:category(D,c) ⇐ dsome:category(D,c) ddg:name(D,n) ⇐ dsome:name(D,n), dsome:diseases(D) ddg:possibleDrug(X,Y) ⇐ dsome:possibleDrug(X,Y), dsome:diseases(X), dmed:drugs(Y)
ddg:Drug(D) ⇐ dmed:drugs(D) ddg:Ingredient(I) ⇐ dmed:ingredients(I) ddg:name(D,n) ⇐ dmed:fullName(D,n), dmed:drugs(D) ddg:name(I,n) ⇐ dmed:label(I,n), dmed:ingredients(I) ddg:indication(D,i) ⇐ dmed:indication(D,i), dmed:drugs(D) ddg:activeIngredient(D,I) ⇐ dmed:activeIngredient(D,I), dmed:drugs(D), dmed:ingredients(I) ddg:genericDrug(D,G) ⇐ dmed:genericDrug(D,G), dmed:drugs(D), dgbank:drugs(G)
ddg:Drug(D) ⇐ dbp-ont:Drug(D) ddg:metabolism(D,m) ⇐ dbpprop:metabolism(D,m), dbp-ont:Drug(D) ddg:pregnancyCategory(D,p) ⇐ dbpprop:pregnancyCategory(D,p), dbp-ont:Drug(D)
ddg:Drug(D) ⇐ sider:drugs(D) ddg:SideEffect(S) ⇐ sider:side_effects(S) ddg:sideEffect(D,S) ⇐ sider:sideEffect(D,S), sider:drugs(D), sider:side_effects(S) ddg:sideEffectName(S,n) ⇐ sider:sideEffectName(S,n), sider:side_effects(S)
ddg:GenericDrug(G) ⇐ dgbank:drugs(G) ddg:affectedOrganism(D,a) ⇐ dgbank:affectedOrganism(D,a), dgbank:drugs(D) ddg:formula(D,f) ⇐ dgbank:chemicalFormula(D,f), dgbank:drugs(D)

Figura 4.6: Mapeamentos de mediação para o mashup de Drogas.

4.3 Processamento de Consultas

O objetivo do sistema de mediação proposto em (PINHEIRO, 2011) é responder consultas SPARQL formuladas pelo usuário em termos da ontologia de domínio. Ao receber uma consulta, o mediador a reescreve em subconsultas que serão executadas pelas diversas fontes de dados que estão sendo integradas. O mediador possui dois módulos principais, conforme representado na Figura 4.7: o módulo responsável por gerar os planos de execução de consultas e o módulo responsável pela execução desses planos, os quais são descritos a seguir.

Módulo de Geração dos planos de consultas federados
A seguir são apresentados os componentes do módulo Gerador de Planos de Execução de Consultas (*Query Execution Plan Generator*):

Catálogo de Metadados armazena dados voiD, mapeamentos e estatísticas sobre os conjuntos de dados registrados.

Data Source Register and Monitor é o responsável pelo registro dos conjuntos de dados que serão acessados através das consultas federadas. O registro de um conjunto de dados permite que metadados voiD (ALEXANDER et al., 2009), possivelmente existentes sobre ele, sejam armazenados no catálogo de metadados. Além disso, esse componente ainda é responsável por monitorar os conjuntos de dados registrados para atualizar suas descrições voiD e estatísticas no catálogo de metadados.

Query Parser gera um plano de consulta canônico sobre a OD, contendo operadores da álgebra SPARQL.

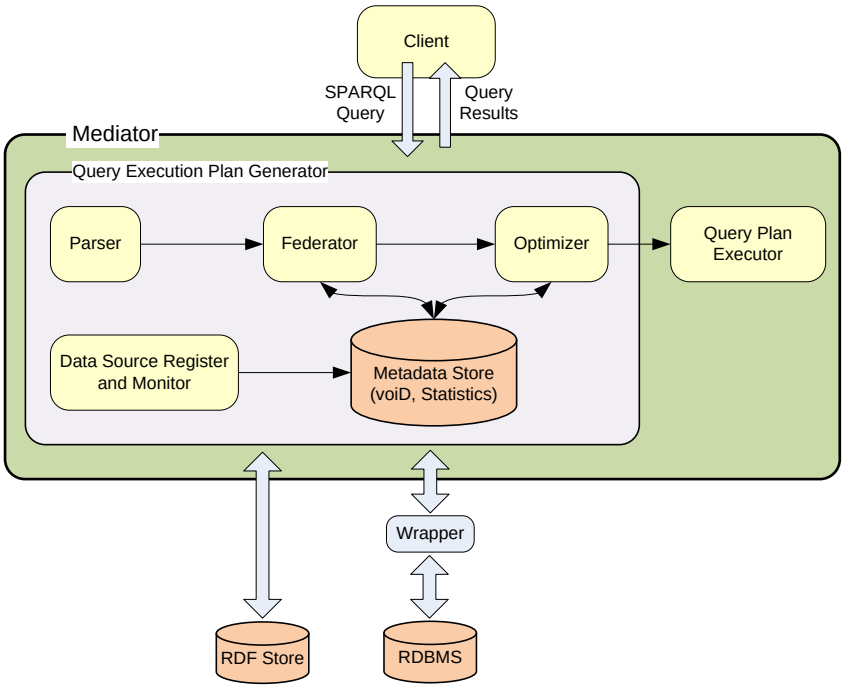

Figura 4.7: Componentes de um mediador

Federator é responsável por reformular o plano de consulta sobre a OD em um plano de consulta federado baseado nas informações do catálogo de metadados.

Optimizer realiza otimizações no plano de consulta federado baseado nas informações colhidas no catálogo de metadados.

A Figura 4.8 mostra uma visão geral do processo para geração do plano de execução proposto em (VIDAL et al., 2011):

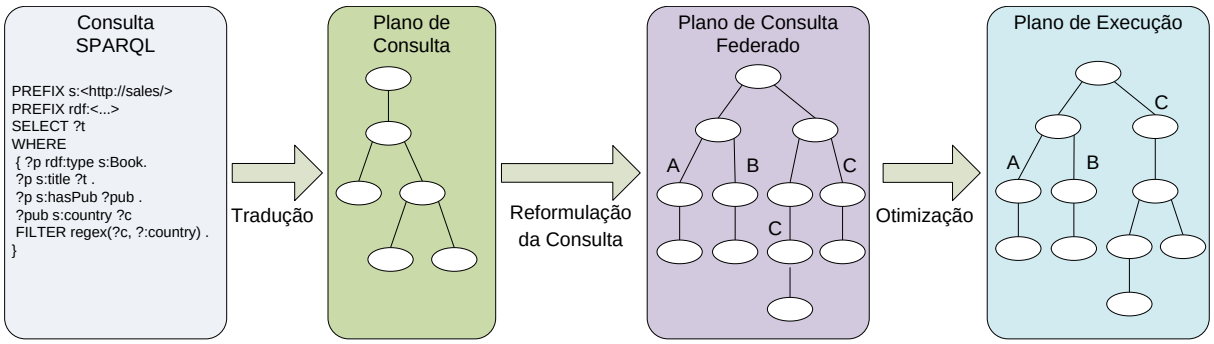

Figura 4.8: Processo para geração do plano de execução da consulta

Módulo de Execução de consultas (QEF-LD)

O módulo *Query Plan Executor* é responsável pela execução de planos de consulta federados sobre a Web de Dados. Este componente também pode ser chamado de QEF-LD por se tratar de uma extensão do *QEF - Query Evaluation Framework* (PORTO et al., 2007) com suporte a Linked Data. O QEF foi estendido para permitir a execução desse tipo de planos de consultas, sendo esta a nossa principal contribuição no contexto da arquitetura com uso de mediadores. O QEF-LD será abordado de forma detalhada no Capítulo 6.

Mostramos a seguir um exemplo de geração de plano de execução.

Exemplo: Considere a consulta Q (vide Figura 4.9) definida sobre a Ontologia de Domínio do mashup D&D, a qual obtém detalhes do medicamento de nome 'Cefadroxil'.

```
PREFIX ddg: <http://arida.lia.ufc.br/diseases_drugs/>

SELECT ?dg_act_ing ?dg_mtb ?dg_frm ?sd_eff
WHERE {
  ?dg   ddg:name 'Cefadroxil' ;
        ddg:formula ?dg_frm;
        ddg:metabolism ?dg_mtb ;
        ddg:sideEffect ?sdef ;
        ddg:activeIngredient ?dgai .
  ?dgai ddg:name ?dg_act_ing .
  ?sdef ddg:sideEffectName ?sd_eff .
}
```

Figura 4.9: Consulta SPARQL parametrizada *Q* sobre a *OD* do mashup D&D.

Inicialmente essa consulta é traduzida para um plano de consulta em álgebra SPARQL que é mostrado na Figura 4.10.

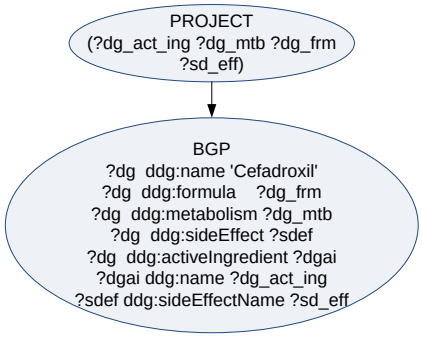

Figura 4.10: Plano de Consulta gerado a partir da consulta *Q*

Depois, esse plano é reformulado para um plano de consulta federado com as operações necessárias para efetuar subconsultas sobre os conjuntos de dados fontes (Figura 4.11). O plano resultante é chamado de plano de consulta federado. A federação de consultas é realizada através da operação SERVICE, especificada na extensão de Federação de Consultas do SPARQL 1.1 (PRUD'HOMMEAUX; BUIL-ARANDA, 2011), que possibilita a determinação do endpoint SPARQL que executará a subconsulta.

Uma otimização possível para esse plano de consulta federado é inverter os operandos da junção para realizar primeiramente a subconsulta sobre o conjunto de dados *dailymed* a fim de selecionar triplas relacionadas à droga definida com o nome 'Cefadroxil' de acordo com o padrão de consulta estabelecido no BGP. Somente depois seria realizada a subconsulta ao *drugbank*, com a variável `?gdg` assumindo os valores obtidos dos resultados da subconsulta sobre o *dailymed*.

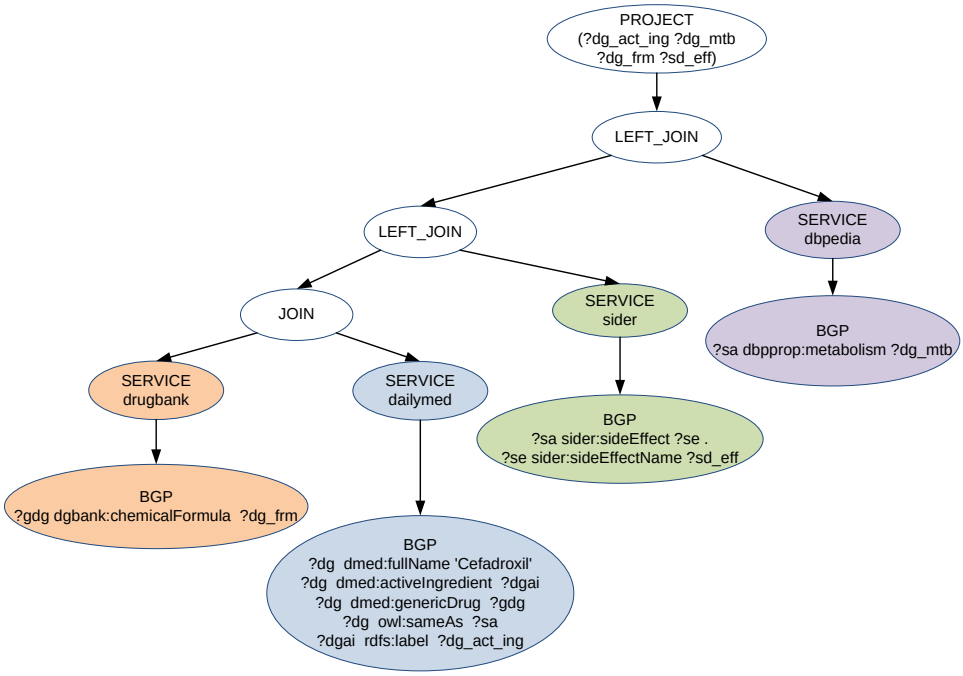

Figura 4.11: Plano de Consulta Federado

Concluído o passo de otimização, o plano otimizado é transformado em um plano de execução no padrão de plano exigido pelo QEF – Query Evaluation Framework (PORTO et al., 2007) – e pronto para ser usado pelo Módulo de Execução (QEF-LD). Detalhes sobre o QEF-LD e sobre a execução do plano serão tratados no capítulo 6. O plano resultante desta última etapa do processo é representado na Figura 4.12 e possui as operações necessárias para que sejam realizadas as subconsultas sobre as fontes de dados relevantes e a construção do resultado final da consulta.

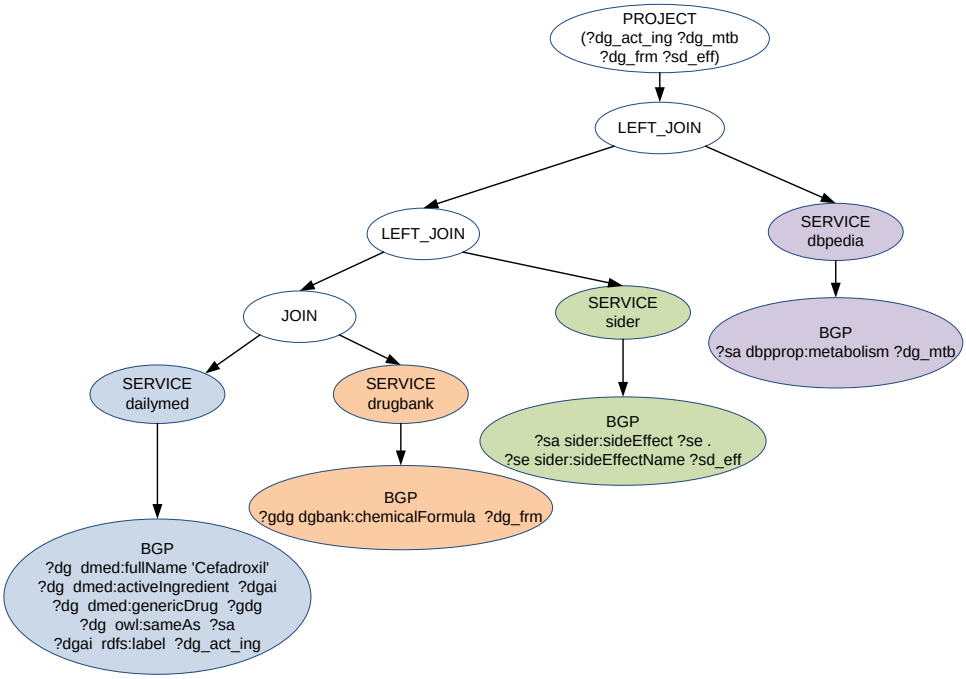

Figura 4.12: Plano de Execução de Consulta

4.4 Conclusões

Este capítulo apresentou uma arquitetura baseada no uso de mediadores para facilitar a construção de Linked Data Mashups. Uma contribuição relevante deste trabalho é a implementação do módulo de execução de consultas federadas (QEF-LD) que pode ser usado tanto na arquitetura de mediadores, como na arquitetura sem uso de mediadores, que será abordada no próximo capítulo. Detalhes sobre o QEF-LD serão tratados no Capítulo 6.

5 ARQUITETURA DE LINKED DATA MASHUPS BASEADA NO USO DE LIDMS

5.1 Introdução

Este capítulo propõe uma arquitetura de *Linked Data Mashups* baseada no uso de *Linked Data Mashup Services* (LIDMS). Apresenta também um enfoque para a especificação conceitual de um LIDMS e a geração automática do LIDMS a partir dessa especificação.

Os LIDMS são serviços web que combinam dados de múltiplas fontes e retornam o resultado no padrão de *Linked Data*. Os dados são extraídos e integrados dinamicamente baseado em parâmetros de entradas. Cada LIDMS tem associado um plano de consulta federado, definido em tempo de projeto mashup, o qual especifica o processo de transformação e integração dos dados das várias fontes.

Esta arquitetura é apropriada quando os padrões de consultas do mashup podem ser definidos a priori, em tempo de projeto do mashup, o que é uma situação comum em aplicações de mashup de dados uma vez que os mashups são normalmente utilizados para necessidades situacionais específicas e demanda curtos períodos de execução. A grande vantagem dessa arquitetura é que não é necessário o uso de um mediador, em tempo de execução, para realizar a geração do plano de consulta federada, que conforme abordamos no capitulo anterior, é um passo bastante complexo, e ainda não existe disponível no mercado um sistema de mediação capaz de realizá-lo de forma eficiente em tempo de execução. A geração dos planos de execução em tempo de projeto possibilita a realização de ajustes precisos, visando a melhoria de seu desempenho.

5.2 Ambiente de Execução de LIDMS

A Figura 5.1 ilustra o ambiente de execução de LIDMS (LEXEN – LIDMS Execution Environment) proposto, o qual realiza o processamento de serviços requisitados a partir de uma URI e a execução eficiente do plano de consulta federado correspondente, realizando assim, o trabalho de integração dos dados em tempo de execução. A URI possui a identificação do plano federado que será executado, bem como parâmetros necessários à execução desse plano. O plano pode possuir parâmetros nomeados que são extraídos da URI e usados para filtrar os resultados da execução do plano. A especificação e implementação do LEXEN é uma importante contribuição deste trabalho. Seus componentes são apresentados a seguir.

O componente *LIDMS Processor* é uma aplicação Web que provê serviços Web REST (FIELDING, 2000). Serviços baseados em REST usam o protocolo HTTP padrão, são fáceis de entender e usar, pois não é necessário utilizar nenhuma API especializada. O *LIDMS Processor* recebe URIs contendo o identificador da consulta, seus parâmetros e o formato de saída desejado. Ao receber a requisição HTTP, ele solicita ao componente *Query Plan Executor* a execução do plano federado correspondente, enviando também os parâmetros necessários à sua execução. Por fim, à medida que os resultados vão sendo obtidos, eles são convertidos para o formato de saída especificado e retornados ao cliente do serviço web. Se os parâmetros

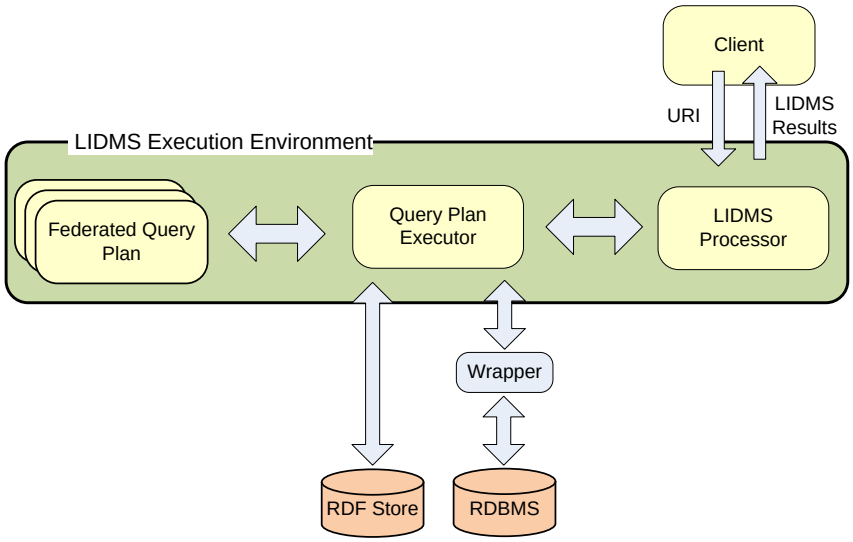

Figura 5.1: Arquitetura baseada no uso de LIDMS

necessários à execução de um plano não forem enviados, o *LIDMS Processor* apresentará um formulário HTML para preenchimento e submissão dos valores desses parâmetros.

O componente *Query Plan Executor* (QEF-LD) é o mesmo componente mencionado na arquitetura de mediadores (Capítulo 4) e estende o *QEF – Query Evaluation Framework* para possibilitar a execução de consultas sobre Linked Data. Ele permite a exploração das ligações que revelam conexões semânticas entre conjuntos de dados. Durante a execução, são realizadas subconsultas sobre os conjuntos de dados relevantes para o plano escolhido. No entanto, seu uso difere nesta arquitetura principalmente pelos dois aspectos citados a seguir: (i) a possibilidade de uso de parâmetros nomeados nos planos de execução, que eram ausentes na arquitetura com uso de mediador; (ii) acesso a planos de execução pré-definidos e armazenados em repositório específico, diferindo dos planos de execução gerados dinamicamente em memória da arquitetura anterior.

Além dos componentes supracitados, o LEXEN também possui um repositório contendo todos os Planos de Execução Federados armazenados em formato XML compatível com o QEF. O QEF-LD executa qualquer um deles de acordo com a demanda imposta pelo *LIDMS Processor*.

A seguir apresentamos o exemplo de um LIDMS gerado para a aplicação de Doenças e Drogas (D&D) descrita no capitulo anterior. Suponha que se deseja construir uma aplicação de mashup de dados que permita um usuário obter informações detalhadas sobre uma droga, e as doenças que ela trata, a partir do nome da droga.

A Figura 5.4 mostra o plano de consulta federado correspondente. Cada plano recebe um identificador único. Note que o plano é similar ao da Figura 4.12, sendo que nome da droga é um parâmetro de entrada. Os valores dos parâmetros de consulta são definidos em tempo de execução por meio de interação do cliente com a interface do LIDMS. No momento da execução, o LEXEN realiza as seguintes atividades: (i) recebimento de URIs enviadas pelos clientes do LIDMS; (ii) processamento de URIs para extrair o identificador do plano de con-

sulta federado que será executado, os parâmetros nomeados e o formato de saída desejado; (iii) carregamento do plano de consulta correspondente ao identificador recebido; (iv) substituição dos parâmetros nomeados pelos seus valores no plano carregado; (v) execução do plano de consulta; (vi) formatação dos resultados para o formato de saída requisitado pelo cliente; (vii) retorno dos resultados formatados.

As atividades (v) a (vii) seguem um fluxo de execução em pipeline. Assim, durante a execução do plano de consulta (atividade v), à medida que os resultados que vão sendo obtidos, eles já vão sendo formatados (atividade vi) e retornados ao cliente (atividade vi), sem que haja necessidade de finalizar uma atividade para poder iniciar a seguinte.

Cache de Planos de Execução de Consultas

O carregamento de planos de execução de consulta a partir dos arquivos XML de *templates* do QEF demanda certo tempo. Para evitar que esse tempo seja gasto antes de cada execução de plano, o LEXEN permite que os planos possam ser pré-carregados para um cache durante a inicialização do LEXEN ou ainda sob demanda, quando a primeira execução de um plano é solicitada. Também é possível alterar um plano de consulta e recarregá-lo para o cache. Ao carregar um plano de execução para o cache, esse plano é analisado para encontrar os parâmetros nomeados que ele usa. Depois disso, os nomes desses parâmetros são armazenados juntamente com o plano para facilitar a verificação do envio dos parâmetros necessários pelo plano a cada solicitação de sua execução. Caso os parâmetros necessários para execução de um plano, não sejam enviados, o LEXEN apresentará um formulário HTML para permitir o preenchimento e submissão dos valores desses parâmetros. Assim, no decorrer da atividade (iii), o LEXEN somente carrega um plano de consulta armazenado em disco, se o plano ainda não estiver armazenado no cache.

Execução de Planos de Consulta com Parâmetros Nomeados

A substituição dos parâmetros nomeados por seus valores, requer alteração do plano de consulta. No entanto, futuras solicitações de execução deverão se basear no plano original e não no plano alterado. Assim, para evitar novo carregamento do plano a partir do disco, optou-se por manter o plano que está no cache intacto, ou seja, sem alterações. Assim, quando for necessário realizar a substituição dos parâmetros por seus valores, o plano que está no cache é clonado, alterado e executado. A execução é realizada sobre o plano clonado e o plano original é mantido intacto para possibilitar futuras gerações de novos clones usando outros valores para os parâmetros nomeados. Após a execução, o plano alterado é removido da memória. No entanto, se o plano de consulta não possuir parâmetros, a execução é realizada sobre o plano original que está no cache.

Suporte a diferentes formatos de saída

Atualmente o LEXEN suporta os formatos de saída XML e JSON que são baseados respectivamente nos documentos *SPARQL Query Results XML Format* (BECKETT; BROEKSTRA, 2008) e *Serializing SPARQL Query Results in JSON* (CLARK; FEIGENBAUM; TORRES, 2008a). Ele também possibilita uma visualização HTML baseada na transformação do XML através de XSLT. Pretende-se ainda adicionar suporte aos formatos de saída para texto plano, bem como para CSV e TSV, tomando como base para estes últimos, o documento *SPARQL 1.1 Query Results CSV and TSV Formats* (SEABORNE, 2011).

5.3 Processo de Geração de LIDMS

O processo de geração do LIDMS envolve uma etapa de modelagem da ontologia de domínio e integração semântica, além de uma etapa adicional especificamente destinada à geração dos LIDMS. As etapas de modelagem da ontologia de domínio e integração semântica possuem os seguintes ingredientes:

1. O esquema conceitual da aplicação é representado por uma ontologia de domínio (OD);
2. Cada fonte de dados é descrita por uma ontologia fonte (OF) disponibilizada como Linked Data e que descreve os dados exportados pela fonte;
3. As correspondências entre a OD e as OFs são especificadas por um conjunto de mapeamentos.

Usando esses ingredientes, o processo de geração de LIDMS consiste de dois passos, que são apresentados a seguir.

Especificação conceitual
Os requisitos de dados do LIDMS são especificados através de uma visão de integração a qual é especificada por uma tripla consiste de uma tripla $< P, O, Q >$, onde (i) P é uma lista de parâmetros de entrada que serão usados em última instância para filtrar os resultados da saída; (ii) O é uma ontologia que descreve o resultado retornado; (iii) Q é uma consulta SPARQL parametrizada, definida sobre a ontologia de domínio OD, conforme mostrado na figura 5.3.

A Figura 5.2 ilustra que cada LIDMS está relacionado através de uma determinada URI, a uma consulta parametrizada sobre a OD. A URI identifica a consulta que será usada e ainda contém os parâmetros que serão usados na execução da consulta.

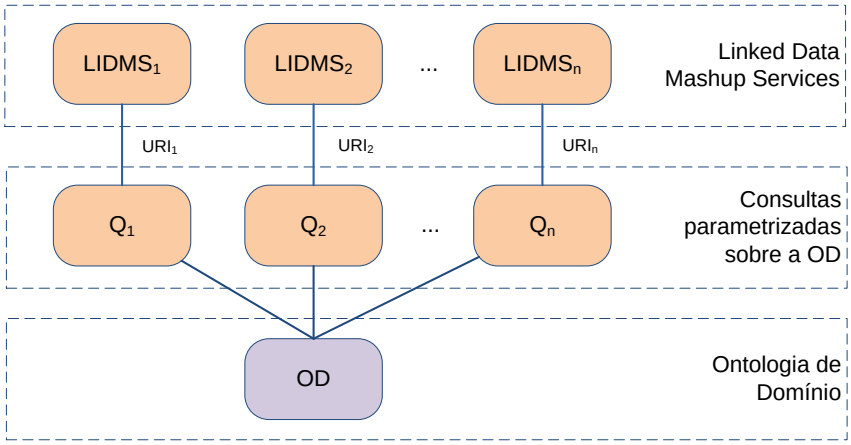

Figura 5.2: LIDMS relacionados a consultas parametrizadas sobre a Ontologia de Domínio

A Figura 5.3 mostra a especificação conceitual do LIDMS *Drug Details*.

Implementação dos LIDMS
O objetivo desse passo é a geração do plano de execução da consulta federada do LIDMS.

LIDMS Drug Details
P : **Input Parameters** drug_name: String
O : **Output description** Drug details: metabolism, formula, side effect names, active ingredients

Q : **Query**
PREFIX ddg: <http://arida.lia.ufc.br/diseases_drugs/>
 	
SELECT ?dg_act_ing ?dg_mtb ?dg_frm ?sd_eff
WHERE {
?dg ddg:name **?:drug_name** ;
ddg:metabolism ?dg_mtb ;
ddg:formula ?dg_frm ;
ddg:sideEffect ?sdef ;
ddg:activeIngredient ?dgai .
?dgai ddg:name ?dg_act_ing .
?sdef ddg:sideEffectName ?sd_eff .
}

Figura 5.3: Especificação conceitual do LIDMS Drug Details

O plano é gerado automaticamente a partir da consulta SPARQL parametrizada sobre a OD, usando o mesmo processo de geração de planos de consulta federados discutido na seção 4.3. A Figura 5.4 mostra uma representação do plano de consulta federado para o LIDMS *Drug Details*.

Após a geração de uma representação do plano de consulta federado em memória, ela é convertida e armazenada em um *template* do QEF em formato XML, que é basicamente um arquivo XML compatível com o formato de plano de execução exigido pelo *QEF – Query Evaluation Framework*. Observe que ao lado esquerdo de cada operador da Figura 5.4 foi colocado um número para identificá-lo. Esse número é usado para identificar cada operador e definir as relações entre produtores e consumidores de resultados de operações em um *template* do QEF. Detalhes sobre o funcionamento do QEF serão abordados no Capítulo 6. O *template* do QEF correspondente ao plano de consulta federado da Figura 5.4 é apresentado da listagem 5.1.

O protocolo SPARQL atualmente não suporta consultas parametrizadas, embora haja proposta para acrescentar este recurso à futuras versões da especificação (MIKHAILOV, 2009). No entanto, algumas ferramentas, como Jena ARQ, Sesame e Virtuoso, implementam esse tipo de consulta. Como há diferença na sintaxe adotada pelas diferentes ferramentas, o ambiente de execução de LIDMS (LEXEN) identifica os parâmetros de um plano usando a sintaxe do Virtuoso por ser mais descritiva, mais fácil de ser identificada e também por facilitar a checagem de erros. Nessa sintaxe, todo parâmetro possui um nome precedido dos caracteres '**?:**'. Imediatamente antes da execução de um plano de consulta federado, o LEXEN substitui os parâmetros nomeados, pelos seus respectivos valores.

```
<?xml version="1.0" encoding="UTF-8"?>
<QEPTemplate xmlns="http://giga03.lncc.br/DIP/WP4/CoDIMS-D"
             xmlns:op="http://giga03.lncc.br/DIP/WP4/CoDIMS-D/Operator"
             xmlns:qep="http://giga03.lncc.br/DIP/WP4/CoDIMS-D/QEP">
  <qep:QEP type="Initial">
```

```
<op:Operator id="1" prod="2" type="">
  <Name>Project</Name>
  <ParameterList>
     <Variables>dg_act_ing, dg_mtb, dg_frm, sd_eff, disease_name, drug_name
     </Variables>
   </ParameterList>
</op:Operator>

<op:Operator id="2" prod="3,4" type="">
  <Name>BindLeftJoin</Name>
</op:Operator>

<op:Operator id="3" prod="5,6" type="">
  <Name>BindLeftJoin</Name>
</op:Operator>

<op:Operator id="4" prod="0" type="Scan" numberTuples="?">
  <Name>Service</Name>
  <ParameterList>
    <DataSourceName>SparqlEndpoint</DataSourceName>
    <ServiceURI>http://dbpedia.org/sparql</ServiceURI>
    <SPARQLQuery>
      <![CDATA[
        PREFIX dbpprop: <http://dbpedia.org/property/>
        select * where {
          ?sa dbpprop:metabolism ?dg_mtb
        }
      ]]>
    </SPARQLQuery>
  </ParameterList>
</op:Operator>

<op:Operator id="5" prod="7,8" type="">
  <Name>SetBindJoin</Name>
</op:Operator>

<op:Operator id="6" prod="0" type="Scan" numberTuples="?">
  <Name>Service</Name>
  <ParameterList>
    <DataSourceName>SparqlEndpoint</DataSourceName>
    <ServiceURI>http://www4.wiwiss.fu-berlin.de/sider/sparql</ServiceURI>
    <SPARQLQuery>
      <![CDATA[
        PREFIX sider: <http://www4.wiwiss.fu-berlin.de/sider/resource/sider/>
        select * where {
          ?sa sider:sideEffect ?se .
          ?se sider:sideEffectName ?sd_eff .
        }
      ]]>
    </SPARQLQuery>
  </ParameterList>
</op:Operator>

<op:Operator id="7" prod="0" type="Scan" numberTuples="?">
  <Name>Service</Name>
  <ParameterList>
    <DataSourceName>SparqlEndpoint</DataSourceName>
    <ServiceURI>http://www4.wiwiss.fu-berlin.de/dailymed/sparql</ServiceURI>
    <SPARQLQuery>
      <![CDATA[
        PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
        PREFIX owl: <http://www.w3.org/2002/07/owl#>
        PREFIX dmed: <http://www4.wiwiss.fu-berlin.de/dailymed/resource/dailymed/>
        select * where {
          ?dg   dmed:fullName         ?:drug_name ;
                dmed:activeIngredient ?dgai ;
```

```
                    dmed:genericDrug       ?gdg ;
                    owl:sameAs             ?sa .
              ?dgai rdfs:label             ?dg_act_ing .
            }
          ]]>
        </SPARQLQuery>
      </ParameterList>
    </op:Operator>

    <op:Operator id="8" prod="0" type="Scan" numberTuples="?">
      <Name>Service</Name>
      <ParameterList>
        <DataSourceName>SparqlEndpoint</DataSourceName>
        <ServiceURI>http://www4.wiwiss.fu-berlin.de/drugbank/sparql</ServiceURI>
        <SPARQLQuery>
          <![CDATA[
            PREFIX dgbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
            select * where {
              ?gdg dgbank:chemicalFormula  ?dg_frm .
            }
          ]]>
        </SPARQLQuery>
      </ParameterList>
    </op:Operator>

        </qep:QEP>
</QEPTemplate>
```

Listagem 5.1: *Template* do QEF com Plano de Execução de Consulta

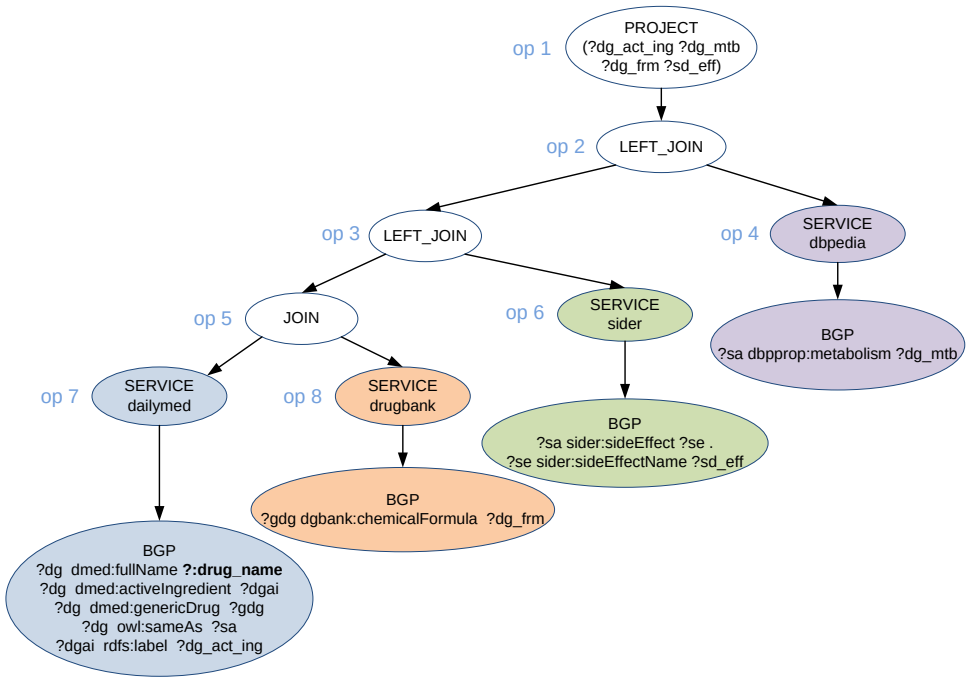

Figura 5.4: Plano de Consulta Federado

5.4 Conclusões

Este capítulo apresentou as características e o funcionamento do ambiente de execução de LIDMS (LEXEN). Este ambiente funciona como um serviço Web para recuperar de

forma eficiente o resultados de planos de consulta federados a partir de uma dada requisição de URI especificando o identificador do plano de execução que se deseja executar, os parâmetros que serão usados nesse plano e o formato de saída resultante. LIDMS representam, portanto, uma alternativa viável e simples ao uso mediadores para a execução eficiente de consultas federadas SPARQL pré-definidas sobre a web de dados. O próximo capítulo aborda os detalhes de funcionamento do QEF-LD que pode ser usado tanto no LIDMS, quanto na arquitetura com uso de mediador. Tratará também dos operadores implementados para tornar o QEF-LD eficiente.

6 EXECUÇÃO DE PLANOS DE CONSULTA FEDERADOS SOBRE A WEB DE DADOS

6.1 Introdução

A principal contribuição deste trabalho é a implementação de um componente capaz de executar de forma eficiente consultas federadas sobre a Web de Dados. Este componente chama-se QEF-LD, por estender o QEF – Query Evaluation Framework (PORTO et al., 2007) – para a execução de planos de consulta federados sobre Linked Data, e pode ser usado em ambas as arquiteturas propostas nos capítulos 4 e 5.

As fontes de dados que usam o modelo RDF são consultadas diretamente. Fontes não RDF necessitam de um *wrapper* para converter os dados do seu modelo nativo para o modelo RDF usado no SWDM. Depois disso, os dados são integrados e manipulados de acordo com as operações definidas no plano de execução para a geração do resultado final da consulta. Esse resultado é, então, serializado em um formato específico usado para retornar resultados de consultas SPARQL, como XML (BECKETT; BROEKSTRA, 2008) ou JSON (CLARK; FEIGENBAUM; TORRES, 2008a), de acordo com o formato solicitado na URL da requisição HTTP. Por fim, o resultado serializado é retornado em uma resposta HTTP ao cliente do *mashup*.

Para melhorar o desempenho das consultas, foram implementados operadores que possibilitam a execução paralela de consultas sobre fontes remotas (paralelismo intraoperador). Além disso, é possível utilizar os resultados de consultas a essas fontes, à medida que vão sendo recebidos, não sendo necessário esperar o recebimento de todos os resultados para somente depois efetuar seu processamento.

Os dados acessados pelo QEF-LD devem ser acessíveis a partir de um *endpoint SPARQL*. Caso essa condição não seja satisfeita, pode-se usar um *wrapper* para prover um *endpoint SPARQL* para os dados, possibilitando a conversão de dados de um modelo específico para o modelo RDF. Exemplos disso são dados de bases relacionais acessíveis através do Servidor D2R (BIZER; CYGANIAK, 2006) ou mesmo planilhas cujos dados podem ser acessados através consultas SPARQL resolvidas pelo *wrapper* XLWrap (LANGEGGER; WÖSS, 2009).

6.2 Mecanismo de Execução dos Planos de Consulta Federados

O QEF-LD é uma extensão ao QEF – *Query Evaluation Framework* (PORTO et al., 2007) para possibilitar a execução de planos de consulta sobre Linked Data. O QEF é um framework que provê um ambiente para definição e execução de planos de consulta. Ele é derivado do CoDIMS – *Configurable Data Integration Middleware System* – que é um *middleware* para a geração de sistemas adaptáveis e configuráveis de integração de dados. O QEF permite a execução de um plano de consulta em um ambiente distribuído, a comunicação entre os componentes de execução de consulta e o acesso a fontes de dados heterogêneas.

Os planos de consulta são representados por um QEP – *Query Execution Plan* (Plano de Execução de Consulta) – e constituídos de operadores algébricos e operadores de controle (OLIVEIRA; PORTO, 2010). As operações de um QEP comunicam-se entre si para a obtenção de um resultado. Um QEP é representado como uma árvore onde os nós são operadores, as folhas são fontes de dados (*data sources*) e as arestas são os relacionamentos entre operadores no modo produtor-consumidor. A estrutura de dados consumida e produzida pelos operadores é chamada de tupla. Os **operadores algébricos** implementam a álgebra de um determinado modelo de dados e agem no conteúdo de uma tupla, realizando processamentos de acordo com uma semântica específica. Os **operadores de controle** são metaoperadores que implementam características de execução associadas ao fluxo de dados.

Os operadores seguem o modelo de iterador (GRAEFE, 1990), implementando as seguintes operações: *open*, *getNext* e *close*. A operação *open* prepara o operador para a produção de dados. Essa preparação consiste em executar a operação *open* de todos os produtores do operador e também em definir os metadados sobre as tuplas que serão produzidas pelo operador, através da operação *setMetadata*. O Algoritmo 1 apresenta a operação *open* usada pelo operador *Operator* do QEF. A operação *setMetadata* (Linha 7) recebe os metadados dos produtores do operador e a partir deles, define os metadados que serão usados para produzir as tuplas resultantes. Usualmente a operação *open* também inicializa algumas variáveis usadas pelo operador.

Algoritmo 1: Operator - `open`

```
   Input: producers
1  i ← 0
2  foreach producer in producers do
3  |  producer.open ()
4  |  metadata[i] ← producer.getMetadata ()
5  |  i++
6  end
7  setMetadata (metadata)
```

A operação *getNext* produz uma tupla sob demanda do consumidor. As tuplas produzidas devem possuir as características definidas pelos metadados do operador. Neste capítulo serão apresentados os algoritmos da operação *getNext* dos operadores implementados como contribuições deste trabalho.

Finalmente, a operação *close* conclui a execução do operador, ao executar *close* em todos os seus produtores, como pode ser observado no Algoritmo 2, usado no operador *Operator* do QEF. Também é comum a liberação de recursos na implementação de *close*.

Algoritmo 2: Operator - `close`

```
   Input: producers
1  foreach producer in producers do
2  |  producer.close ()
3  end
```

A chamada do operador localizado na raiz da árvore é propagada aos operadores filhos até alcançar as folhas (fontes de dados). A operação *getNext* requisita a produção de uma tupla a ser consumida por um consumidor dentro da cadeia de execução. O resultado é uma execução *pipeline* de operadores sincronizados pela chamada da operação *getNext* entre pares de operadores produtor-consumidor. Esse modo de execução *pipeline* permite a produção de resultados assim que a primeira tupla chega ao operador raiz.

O QEF possibilita a criação de fontes de dados (*data sources*) para realizar o acesso transparente a dados heterogêneos armazenados em diferentes localizações. Através da implementação da interface *DataSource*, que funciona como um *wrapper*, é possível abstrair o formato e a localização dos dados, facilitando o acesso a eles.

Cada plano de execução no QEF é definido por um *template* que é um arquivo XML composto de uma lista de operadores, onde cada operador é definido por um *id*, um nome, uma lista de produtores e uma lista de parâmetros.

O QEF suporta a distribuição da execução de consultas em um ambiente de grid. A distribuição e o paralelismo proporcionado por essa arquitetura visam a diminuição do tempo de execução das consultas. No entanto, o presente trabalho usa um cenário de uso do QEF em que o processamento das consultas sobre fontes de dados distribuídas ocorre de forma centralizada em um único host, e o paralelismo ocorre através do uso de (*threads*).

6.3 Implementação de extensões do QEF para lidar com a álgebra SPARQL

Uma das contribuições deste trabalho foi a implementação de extensões do QEF para permitir a execução de consultas federadas parametrizadas usando a álgebra SPARQL. Buscou-se melhorar o desempenho especialmente através do uso de *threads* para possibilitar o **paralelismo intraoperador** na obtenção de resultados dos produtores; e também através da **redução do número de chamadas a *endpoints* remotos**. Em alguns casos também foi possível **limitar a quantidade de resultados obtidos** através da ligação (*binding*) de variáveis. Os operadores de junção foram os principais alvos dessas melhorias de desempenho desde que usualmente causam maior impacto no custo das consultas. A extensões implementadas são abordadas a seguir:

SPARQL Endpoint Data Source
Uma fonte de dados específica para acesso a *endpoints SPARQL* foi criada para possibilitar a execução de consultas SPARQL a conjuntos de dados remotos. A implementação usa internamente o *framework* Jena 2.6.11 para obter os dados de um determinado *endpoint SPARQL* e convertê-los para a representação de dados usada internamente no QEF (tupla). A fonte de dados ainda implementa suporte ao uso de parâmetros nomeados em consultas.

Operador *Service*
O operador *Service* é um operador de acesso a fontes de dados do tipo *endpoint SPARQL*. Ele recebe como parâmetros o nome da fonte de dados responsável pela execução da consulta (*DataSourceName*), a string da consulta SPARQL (*SPARQLQuery*) e a URI do *endpoint* SPARQL (*ServiceURI*) onde a consulta será executada. Os dois últimos parâmetros são repassados à

fonte de dados definida no primeiro parâmetro. A string da consulta SPARQL pode conter parâmetros nomeados que serão substituídos na string por seus respectivos valores. Os nomes dos parâmetros são precedidos pelos caracteres '?:'. A Seção 5.2 aborda mais detalhes sobre essa sintaxe adotada para nomear parâmetros. A Figura 6.1 apresenta um trecho de um *template* do QEF com um exemplo de definição de um operador *Service*, onde o parâmetro nomeado `?:dsname` será substituído pelo seu valor durante a execução da consulta.

```
<op:Operator id="1" prod="0" type="Scan" numberTuples="?">
<Name>Service</Name>
<ParameterList>
    <DataSourceName>SparqlEndpoint</DataSourceName>
    <ServiceURI>http://www4.wiwiss.fu-berlin.de/diseasome/sparql</ServiceURI>
    <SPARQLQuery>
    <![CDATA[
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX dsome: <http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/>

    SELECT ?ds ?dsn
    WHERE {
         ?ds dsome:name ?dsn .
         FILTER regex(?dsn, ?:dsname, "i")
    }
    ORDER BY ?dsn
    LIMIT 100
    ]]>
    </SPARQLQuery>
</ParameterList>
</op:Operator>
```

Figura 6.1: Definição de um operador Service em *template* do QEF

Operador *Project*

O operador *Project* é usado para definir as variáveis SPARQL cujos valores serão projetados nos resultados da consulta.

Operador *BindJoin*

O operador *BindJoin* obtém o resultado da junção entre os seus produtores. Para cada tupla obtida do produtor esquerdo da junção, são recuperadas as tuplas do produtor direito, relacionadas à tupla obtida. O predicado de junção é dado pelas variáveis comuns entre os produtores do *join*. Os valores dessas variáveis comuns obtidos do produtor esquerdo do *join* são usados para fazer o (*bind*) das variáveis existentes na(s) consulta(s) realizadas pelo produtor direito do *join*. Depois disso, são retornadas as tuplas resultantes da junção entre a tupla obtida do produtor esquerdo e as tuplas do produtor direito do *join*. Os passos descritos são repetidos para cada uma das demais tuplas obtidas pelo produtor esquerdo da junção.

Operador *BindLeftJoin*

O operador *BindLeftJoin* obtém o resultado da junção esquerda (*left join*) entre os produtores da junção. O operador realiza praticamente os mesmos passos do *BindJoin*, exceto pelo fato de que se não for recuperada tupla do produtor direito, correspondente a uma tupla gerada pelo produtor esquerdo do join, a tupla do produtor esquerdo será considerada na solução. A essa tupla ainda é necessário incluir as variáveis presentes no produtor direito, mas ausentes no produtor esquerdo, com valores nulos para completar a tupla resultante.

Operador *SetBindJoin*
O operador *SetBindJoin* assim como o operador *BindJoin* também obtém o resultado da junção entre os seus produtores. No entanto, ele difere do operador *BindJoin* por agrupar os resultados do produtor esquerdo em um conjunto (*set*) cujo tamanho pode ser configurado no plano de execução. Depois, são obtidas as tuplas do produtor direito do join que possuem relação com as tuplas do conjunto. Por fim, é realizada a junção entre as tuplas do conjunto e as tuplas obtidas do produtor direito do join, cujas variáveis comuns entre ambos os produtores do join possuem o mesmo valor. Além disso, pode-se habilitar o uso de threads para paralelizar a obtenção de resultados, permitindo que novos conjuntos do lado esquerdo sejam formados para obtenção de resultados, independente de conjuntos anteriores já terem sido completamente processados para obtenção de resultados. O *BindJoin* funciona de modo semelhante a um *SetBindJoin*, que não usa *threads* e onde o conjunto (*set*) de resultados do produtor esquerdo é configurado para ter somente um único elemento.

Operador *Union*
O operador *Union* permite obter o resultado da união das tuplas obtidas a partir de múltiplos produtores. Pode-se habilitar o uso de *threads* para que os resultados dos produtores sejam obtidos de forma paralela para geração do resultado final da união.

6.4 Algoritmo BindJoin / BindLeftJoin

O Algoritmo 3 foi desenvolvido para realizar a junção e junção à esquerda, a partir da ligação (*bind*) das variáveis comuns entre os produtores da junção. Assim, para cada tupla obtida do produtor esquerdo da junção, são obtidas as tuplas correspondentes do produtor direito.

A variável booleana *leftJoin* quando possui o valor `true`, define o uso do operador *LeftJoin*. Desse modo, mesmo não havendo tupla gerada pelo produtor direito do *LeftJoin*, uma tupla resultante será gerada pelo método *fillTupleWithNullToRightProducerVariables* (Linha 12 do Algoritmo 3) com o conteúdo da tupla obtida pelo produtor esquerdo, acrescido de valores nulos para as variáveis usadas somente no produtor direito do *join*.

O método `cloneAndReformulate` (Linha 6) clona e altera o produtor direito do join, reformulando as consultas existentes nesse produtor, de modo que os resultados obtidos a partir delas somente recupere resultados relacionados à tupla gerada pelo produtor esquerdo. Para exemplificar, suponhamos que o produtor esquerdo do join tenha retornado a tupla ilustrada da Figura 6.2 e que o produtor direito do join possua a consulta SPARQL mostrada na Figura 6.3.

```
( ?dblp_researcher = <http://dblp.l3s.de/d2r/resource/authors/Marco_A._Casanova>,
  ?publication = <http://dblp.l3s.de/d2r/resource/publications/journals/jcss/CasanovaFP84> )
```

Figura 6.2: Exemplo de tupla gerada pelo produtor esquerdo do join

Ao executar o método `cloneAndReformulate`, um novo produtor direito será cri-

Algoritmo 3: BindJoin / BindLeftJoin - `getNext`

```
   Input: leftProducer, rightProducer, emptyRightTuple, leftJoin
   Output: tuple
 1 if emptyRightTuple then
 2 |   leftTuple ← leftProducer.getNext ()
 3 |   if leftTuple = null then
 4 |   |   return null
 5 |   end
 6 |   changedRightProducer ← rightProducer.cloneAndReformulate (leftTuple)
 7 end
 8 rightTuple ← changedRightProducer.getNext ()
 9 tuple ← leftTuple.clone ()
10 if rightTuple = null then
11 |   if leftJoin and emptyRightTuple then
12 |   |   fillTupleWithNullToRightProducerVariables (tuple)
13 |   |   emptyRightTuple ← true
14 |   else
15 |   |   emptyRightTuple ← true
16 |   |   return getNext ()
17 |   end
18 else
19 |   tuple ← createJoinedTuple (leftTuple, rightTuple)
20 |   emptyRightTuple ← false
21 end
22 return tuple
```

```
prefix dc:   <http://purl.org/dc/elements/1.1/>

SELECT  * WHERE {
  ?publication dc:creator ?dblp_researcher .
  ?publication dc:title ?pub_title
}
```

Figura 6.3: Exemplo de consulta SPARQL existente no produtor direito do join

ado. Ele terá uma consulta SPARQL reformulada (Figura 6.4) em relação à consulta original. Essa consulta retornará somente resultados relacionados à tupla gerada pelo produtor esquerdo do join. A reformulação consiste em aplicar um filtro à consulta SPARQL original de modo a obter somente as tuplas que efetivamente produzirão algum resultado final na junção com as tuplas geradas pelo produtor esquerdo. O método *bindVariables* da classe *ch.epfl.codimsd.qeef.sparql.JoinQueryManipulation* do QEF é usado para realizar a reformulação.

```
prefix dc:   <http://purl.org/dc/elements/1.1/>

SELECT  * WHERE {
  ?publication dc:creator ?dblp_researcher .
  ?publication dc:title ?pub_title
  FILTER ( ?dblp_researcher = <http://dblp.l3s.de/d2r/resource/authors/Marco_A._Casanova> &&
    ?publication = <http://dblp.l3s.de/d2r/resource/publications/journals/jcss/CasanovaFP84> )
}
```

Figura 6.4: Exemplo de consulta SPARQL do produtor direito do join reformulada

Além da estratégia de reformulação adotada que usa filtro e uma conjunção de igualdades, ainda seria possível obter os mesmos resultados usando as seguintes estratégias: (i) `BINDINGS`; e (ii) Filtro usando o operador `IN`. A Figura 6.5 ilustra o uso dessas estratégias para o exemplo de reformulação presentado anteriormente. A estratégia (i) não foi usada porque o recurso `BINDINGS` somente está presente a partir da especificação 1.1 de SPARQL e não é suportado por grande parte dos endpoints SPARQL atualmente disponíveis. Já a estratégia (ii) foi descartada, pois o operador `IN` somente possibilita o uso de uma única variável e também porque ele, assim como `BINDINGS` somente está disponível a partir do SPARQL 1.1.

(a) Reformulação usando BINDINGS

```
prefix dc:   <http://purl.org/dc/elements/1.1/>

SELECT  * WHERE {
  ?publication dc:creator ?dblp_researcher .
  ?publication dc:title ?pub_title
}
BINDINGS ?dblp_researcher ?publication {
 ( <http://dblp.l3s.de/d2r/resource/authors/Marco_A._Casanova>
   <http://dblp.l3s.de/d2r/resource/publications/journals/jcss/CasanovaFP84> )
}
```

(b) Reformulação usando Filtro com o operador IN

```
prefix dc:   <http://purl.org/dc/elements/1.1/>

SELECT  * WHERE {
  ?publication dc:creator ?dblp_researcher .
  ?publication dc:title ?pub_title
  FILTER (
    ?dblp_researcher IN ( <http://dblp.l3s.de/d2r/resource/authors/Marco_A._Casanova>)
  )
}
```

Figura 6.5: Exemplo de consultas usando diferentes estratégias de reformulação

6.5 Algoritmo SetBindJoin

O algoritmo SetBindJoin possibilita que os resultados da junção sejam obtidos a partir do processamento paralelo de conjuntos (*sets*) de tuplas gerados pelo produtor esquerdo da junção. O agrupamento das tuplas obtidas do produtor esquerdo da junção em conjuntos, permite a redução do número de consultas remotas a *endpoints* SPARQL relacionados ao produtor direito da junção. Além disso, também funciona como limitador do número de tuplas obtidas, visto que o *binding* das variáveis comuns usadas nos produtores ocasionará a formulação de uma consulta com uma seletividade menor e, portanto, mais restritiva.

O processamento de cada um desses conjuntos pode ser resumidamente dividido nas seguintes etapas:

(i) Formar um conjunto *S* de tuplas obtidas a partir do produtor esquerdo da junção.
(ii) Obter as tuplas do produtor direito da junção que se relacionam com as tuplas do conjunto *S*.
(iii) Gerar as tuplas resultantes a partir da junção entre as tuplas do conjunto *S* e as tuplas obtidas do produtor direito.

As etapas citadas são detalhadas a seguir.

(i) Formar um conjunto *S* de tuplas obtidas a partir do produtor esquerdo da junção.
O algoritmo SetBindJoin (Algoritmos 4 e 5) agrupa as tuplas obtidas do produtor esquerdo da junção, em conjuntos (*sets*) (Linhas 6 a 16). Os conjuntos possuem um tamanho máximo de tuplas previamente configurado no operador *SetBindJoin* do plano de consulta.

(ii) Obter as tuplas do produtor direito da junção que se relacionam com as tuplas do conjunto *S*.

À medida que o conjunto *S* é preenchido com a quantidade de tuplas estipulada em sua configuração, o método *cloneAndReformulate* é executado sobre o produtor direito da junção (Linha 17), para cloná-lo e alterá-lo de modo que a execução do método *getNext*, somente obtenha tuplas relacionadas às tuplas do conjunto *S*. Assim, todas as tuplas obtidas a partir do produtor direito da junção, efetivamente participarão da construção das tuplas resultantes. A alteração do produtor direito consiste em reformular as consultas existentes nele, de modo a fazerem a ligação (*binding*) dos valores das variáveis comuns entre os produtores da junção. Esse valores são extraídos das tuplas do conjunto *S*. A reformulação consiste em alterar as consultas existentes no produtor direito da junção para realizarem o *binding* dos valores das variáveis a partir do uso da operação UNION sobre a mesma estratégia de *binding* adotada no operador *BindJoin* (Seção 6.4). A título de exemplo, suponhamos que o conjunto de tuplas obtido do produtor esquerdo do join possua as tuplas ilustradas da Figura 6.6 e que o produtor direito do join possua a consulta SPARQL mostrada na Figura 6.3.

```
( ?dblp_researcher = <http://dblp.l3s.de/d2r/resource/authors/Marco_A._Casanova>,
  ?publication = <http://dblp.l3s.de/d2r/resource/publications/journals/jcss/CasanovaFP84> ),
( ?dblp_researcher = <http://dblp.l3s.de/d2r/resource/authors/Vania_Maria_Ponte_Vidal>,
  ?publication = <http://dblp.l3s.de/d2r/resource/publications/conf/pods/CasanovaV83> ),
( ?dblp_researcher = <http://dblp.l3s.de/d2r/resource/authors/Jose_Antonio_Fernandes_de_Macedo>,
  ?publication = <http://dblp.l3s.de/d2r/resource/publications/journals/ijbdcn/VidalMPCP11> ),
```

Figura 6.6: Exemplo de conjunto de tuplas geradas pelo produtor esquerdo do join

A Figura 6.7 mostra mesma consulta após sua reformulação pelo método *cloneAndReformulate*. Observa-se o uso de uma operação UNION para cada tupla proveniente do conjunto de resultados do produtor esquerdo da junção, exceto para a primeira tupla. Assim, para um conjunto de *n* tuplas, haverá $(n-1)$ operações UNION após a reformulação.

Outras estratégias de reformulação foram testadas, mas não se mostraram viáveis por apresentarem alguma incompatibilidade com grande parte dos Endpoints SPARQL atualmente existentes ou por mostrarem desempenho inferior ao obtido na estratégia adotada. A Figura 6.8 ilustra a reformulação através do uso da operação BINDINGS proposta na versão 1.1 da especificação de SPARQL.

A Figura 6.9 mostra o uso de FILTER aliado a disjunções de conjunções. Esta última estratégia mostrou-se muito mais lenta que a estratégia adotada. Todas essas estratégias são equivalentes para obtenção dos mesmos resultados. No entanto, as atuais implementações de SPARQL ainda não são capazes de gerar planos sintaticamente equivalentes e otimizados em álgebra SPARQL para as consultas apresentadas. Em SQL, esse problema foi sendo resolvido ao longo do tempo e, atualmente, consultas equivalentes são em grande parte executadas por um mesmo plano de execução. Espera-se que essa evolução também ocorra com as implementações de SPARQL. A estratégia de reformulação adotada resulta em planos de execução que fazem o uso de índices, evitando a busca sequencial nos processadores de consulta testados (Jena ARQ, Sesame e Virtuoso).

Todas as tuplas obtidas pelo produtor esquerdo da junção do conjunto *S* são armazenadas em uma tabela *Hash* (*hash table*) chamada *leftTupleHashTable* (Linhas 4, 8, 11 e 17) que

```
prefix dc:   <http://purl.org/dc/elements/1.1/>

SELECT * WHERE {
  { ?publication dc:creator ?dblp_researcher .
    ?publication dc:title ?pub_title
    FILTER (
     ?dblp_researcher = <http://dblp.l3s.de/d2r/resource/authors/Marco_A._Casanova> &&
     ?publication = <http://dblp.l3s.de/d2r/resource/publications/journals/jcss/CasanovaFP84>
    )
  }
  UNION
  { ?publication dc:creator ?dblp_researcher .
    ?publication dc:title ?pub_title
    FILTER (
     ?dblp_researcher = <http://dblp.l3s.de/d2r/resource/authors/Vania_Maria_Ponte_Vidal> &&
     ?publication = <http://dblp.l3s.de/d2r/resource/publications/conf/pods/CasanovaV83>
    )
  }
  UNION
  { ?publication dc:creator ?dblp_researcher .
    ?publication dc:title ?pub_title
    FILTER (
     ?dblp_researcher = <http://dblp.l3s.de/d2r/resource/authors/Jose_Antonio_Fernandes_de_Macedo> &&
     ?publication = <http://dblp.l3s.de/d2r/resource/publications/journals/ijbdcn/VidalMPCP11>
    )
  }
}
```

Figura 6.7: Exemplo de consulta SPARQL do produtor direito do join reformulada

possui como chave, uma representação dos valores das variáveis comuns entre os produtores da junção; e como valor, uma lista de tuplas que possuem aquela chave.

(iii) Gerar as tuplas resultantes a partir da junção entre as tuplas do conjunto *S* e as tuplas obtidas do produtor direito.
A partir da chave de cada tupla proveniente do produtor direito da junção, obtém-se da *leftTupleHashTable* a lista com todas as tuplas obtidas à esquerda que possuem a mesma chave. Depois disso, percorre-se essa lista para efetuar a junção de cada um de seus elementos com o elemento obtido à direita, produzindo-se, assim, as tuplas resultantes da operação (Linhas 20 a 29).

As tuplas resultantes de todos os conjuntos processados em paralelo são armazenadas em uma única fila bloqueante ligada (*linked blocking queue*) chamada de *resultBuffer*. O método *take* da fila *resultBuffer* (Linha 7 do Algoritmo 4) obtém e remove o elemento do início da fila, caso a fila não esteja vazia. Se a fila estiver vazia, o método *take* entra em estado de espera até que algum novo elemento seja adicionado. Para inserir um elemento no final da fila, utiliza-se o método *put* (Linhas 27 e 33 do Algoritmo 5). O método *put* entra em estado de espera caso não haja espaço disponível para inserção de um novo elemento na fila. Assim, que o espaço for disponibilizado, a fila sai do estado de espera e permite a inserção de novos elementos.

O elemento *END_TOKEN* é usado para sinalizar o final do processamento de todas as tuplas. Ele é inserido após a adição da última tupla resultante. Um contador de conjuntos processados em paralelo (*leftProducerSetCounter*) foi criado para facilitar a identificação final do processamento de todas as tuplas. Ele é incrementado no início do processamento de cada

```
prefix dc:   <http://purl.org/dc/elements/1.1/>

SELECT  * WHERE {
  ?publication dc:creator ?dblp_researcher .
  ?publication dc:title ?pub_title
}
BINDINGS ?dblp_researcher ?publication {
  ( <http://dblp.l3s.de/d2r/resource/authors/Marco_A._Casanova>
   <http://dblp.l3s.de/d2r/resource/publications/journals/jcss/CasanovaFP84> )
  ( <http://dblp.l3s.de/d2r/resource/authors/Vania_Maria_Ponte_Vidal>
   <http://dblp.l3s.de/d2r/resource/publications/conf/pods/CasanovaV83> )
  ( <http://dblp.l3s.de/d2r/resource/authors/Jose_Antonio_Fernandes_de_Macedo>
   <http://dblp.l3s.de/d2r/resource/publications/journals/ijbdcn/VidalMPCP11> )
}
```

Figura 6.8: Exemplo de consulta reformulada usando BINDINGS

```
prefix dc:   <http://purl.org/dc/elements/1.1/>

SELECT  * WHERE {
 ?publication dc:creator ?dblp_researcher .
 ?publication dc:title ?pub_title
 FILTER (
   ( ?dblp_researcher = <http://dblp.l3s.de/d2r/resource/authors/Marco_A._Casanova> &&
     ?publication = <http://dblp.l3s.de/d2r/resource/publications/journals/jcss/CasanovaFP84> ) ||
   ( ?dblp_researcher = <http://dblp.l3s.de/d2r/resource/authors/Vania_Maria_Ponte_Vidal> &&
     ?publication = <http://dblp.l3s.de/d2r/resource/publications/conf/pods/CasanovaV83> ) ||
   ( ?dblp_researcher = <http://dblp.l3s.de/d2r/resource/authors/Jose_Antonio_Fernandes_de_Macedo> &&
     ?publication = <http://dblp.l3s.de/d2r/resource/publications/journals/ijbdcn/VidalMPCP11> )
 )
}
```

Figura 6.9: Exemplo de consulta reformulada usando FILTER e disjunção de conjunções

conjunto e decrementado após o término do processamento de cada conjunto. Desse modo, quando seu valor é zero e não há mais tuplas sendo produzidas pelo produtor esquerdo da junção (Linha 32 do Algoritmo 5), não há mais nada a processar, e o *END_TOKEN* pode ser inserido (Linha 33).

O operador SetBindJoin implementado no QEF é configurável a partir de parâmetros definidos no plano de execução. Esses parâmetros permitem (i) a determinação do tamanho máximo do set utilizado; (ii) a habilitação ou não do uso de threads e; (iii) a determinação do número máximo de threads que podem ser executadas simultaneamente, que termina sendo definido pelo número máximo de conjuntos que o produtor esquerdo da junção (maxNumberOfLeftProducerSets) pode executar de forma paralela. A linha 18 do algoritmo 5 implementa essa limitação. Este último parâmetro demonstrou-se extremamente importante para evitar o acúmulo de threads não processadas e, consequentemente, a diminuição da quantidade de conexões abertas em espera. Também é possível definir essa quantidade máxima de threads simultâneas como ilimitada. No entanto, deixar esse valor ilimitado pode inviabilizar a execução de várias consultas, tendo em vista que o excessivo número de sockets abertos pode ocasionar uma exceção capaz de encerrar a execução da consulta.

Algoritmo 4: SetBindJoin - `getNext`

```
Input: leftProducer, rightProducer, leftTuplesSetSize, resultBuffer, processStarted,
       maxNumberOfLeftProducerSets
Output: tuple
1  if not processStarted then
2      processStarted ← true
3      parallel
4          processTuples (leftProducer, rightProducer, leftTuplesSetSize,
           resultBuffer, maxNumberOfLeftProducerSets)
5      end
6  end
7  tuple ← resultBuffer.take ()
8  if tuple = END_TOKEN then
9      tuple ← null
10 end
11 return tuple
```

Algoritmo 5: SetBindJoin - `processTuples`

```
Input: leftProducer, rightProducer, leftTuplesSetSize, resultBuffer,
       maxNumberOfLeftProducerSets
1  leftTuple ← leftProducer.getNext ()
2  leftProducerSetCounter ← 0
3  while leftTuple ≠ null do
4      leftTuplesHashTable ← createHashtable ()
5      numberOfLeftTuples ← 0
6      while (numberOfLeftTuples < leftTuplesSetSize) and leftTuple ≠ null do
7          key ← getKeyBasedOnSharedVars (leftTuple)
8          leftList ← leftTuplesHashTable.get (key)
9          if leftList = null then
10             leftList ← createList ()
11             leftTuplesHashTable.put (key, leftList)
12         end
13         leftList.add (leftTuple)
14         leftTuple ← leftProducer.getNext ()
15         numberOfLeftTuples++
16     end
17     changedRightProducer ← rightProducer.cloneAndReformulate
       (leftTuplesHashTable)
18     Wait until leftProducerSetCounter < maxNumberOfLeftProducerSets
19     parallel
20         leftProducerSetCounter++
21         rightTuple ← changedRightProducer.getNext ()
22         while rightTuple ≠ null do
23             key ← getKeyBasedOnSharedVars (rightTuple)
24             leftTuplesList ← leftTuplesMap.get (key)
25             foreach leftTuple in leftTuplesList do
26                 tuple ← join (leftTuple, rightTuple)
27                 resultBuffer.put (tuple)
28             end
29             rightTuple ← changedRightProducer.getNext ()
30         end
31         leftProducerSetCounter−−
32         if leftTuple = null and leftProducerSetCounter = 0 then
33             resultBuffer.put (END_TOKEN)
34         end
35     end
36 end
```

6.6 Algoritmo Union

O algoritmo Union (Algoritmo 6) realiza a união de tuplas de múltiplos produtores em paralelo. Uma linha de execução (*thread*) é criada para obter as tuplas de cada produtor e armazená-las em uma única fila bloqueante ligada (*linked blocking queue*) chamada de *resultBuffer*. O método *take* da fila *resultBuffer* (Linha 19 do Algoritmo 6) obtém e remove o elemento do início da fila, caso a fila não esteja vazia. Se a fila estiver vazia, o método *take* entra em estado de espera até que algum novo elemento seja adicionado. Para inserir um elemento no final da fila, utiliza-se o método *put* (Linhas 9 e 14 do Algoritmo 6). O método *put* entra em estado de espera caso não haja espaço disponível para inserção de um novo elemento na fila. Assim, que o espaço for disponibilizado, a fila sai do estado de espera e permite a inserção de novos elementos.

O elemento *END_TOKEN* é usado para sinalizar o final do processamento de todas as tuplas. Ele é inserido após a adição da última tupla resultante. Um contador de produtores processados em paralelo (*producersCounter*) foi criado para facilitar a identificação final do processamento de todas as tuplas. Ele é incrementado no início do processamento de cada produtor e decrementado após o término do processamento. Desse modo, quando seu valor é zero (Linha 13 do Algoritmo 6), não há mais nada a processar, e o *END_TOKEN* pode ser inserido (Linha 14 do Algoritmo 6).

Algoritmo 6: Union - `getNext`

```
   Input: producers, processStarted, resultBuffer
   Output: tuple
 1 if not processStarted then
 2     processStarted ← true
 3     producersCounter ← 0
 4     for i = 0 to producers.size() − 1 do
 5         parallel
 6             producersCounter++
 7             prodTuple ← producers[i].getNext()
 8             while prodTuple ≠ null do
 9                 resultBuffer.put(prodTuple)
10                 tuple ← producers[i].getNext()
11             end
12             producersCounter−−
13             if producersCounter = 0 then
14                 resultBuffer.put(END_TOKEN)
15             end
16         end
17     end
18 end
19 tuple ← resultBuffer.take()
20 if tuple = END_TOKEN then
21     tuple ← null
22 end
23 return tuple
```

6.7 Conclusões

Este capítulo tratou do processamento de planos de consulta federados sobre a web de dados usando o *QEF – Query Evaluation Framework*. Uma especial atenção foi dada à implementação de operadores de junção e união capazes de explorar o paralelismo intraoperador, redução do número de chamadas remotas e limitação do número de resultados obtidos. Também houve uma busca por manter a compatibilidade com Endpoints SPARQL 1.0. O capítulo a seguir avalia através de experimentos a viabilidade do uso desses operadores, comparados com outras estratégias de execução de consultas federadas sobre Linked Data.

7 EXPERIMENTOS E RESULTADOS

7.1 Introdução

A avaliação dos resultados foi baseada na análise comparativa do desempenho de oito consultas sobre *Linked Data*, com variações nos seguintes itens: paralelismo; operadores usados; uso de diferentes implementações e parametrizações dos operadores implementados.

As ferramentas Jena, Sesame, FedX e QEF foram usadas para execução das consultas, sendo todas elas implementadas na linguagem Java. Jena, Sesame e FedX executam as consultas SPARQL federadas diretamente. Já no caso do QEF, foi necessário converter cada consulta SPARQL federada em um plano de consulta correspondente compatível com o QEF. Atualmente essa conversão ainda é manual, mas pretende-se automatizar esse processo em trabalhos futuros (Seção 8.2).

7.2 Execução dos experimentos

Foram criadas três consultas (Q1, Q2, e Q3) para avaliar as estratégias de Junção das ferramentas usadas nos experimentos. Outras três consultas (Q4, Q5 e Q6) avaliam a junção à esquerda. Finalmente, as consultas Q7 e Q8 são utilizadas para comparar desempenhos relacionados à operação de União.

Foram realizados 10 ciclos de execução para cada consulta. Nas consultas contendo operadores configuráveis, cada configuração de uma mesma consulta também teve 10 ciclos de execução. Um exemplo de operador configurável é o SetBindJoin, que permite determinar o tamanho máximo dos sets, habilitar ou não o uso de threads e limitar o número máximo de threads simultâneas. Cada ciclo de execução de consulta envolveu duas execuções, que neste trabalho foram denominadas Execução 1 e Execução 2. A primeira execução (Execução 1) normalmente tem um desempenho inferior devido à inicialização da máquina virtual Java que prepara e aloca os recursos necessários para a execução da aplicação. Já a segunda execução (Execução 2) ocorre na mesma instância da máquina virtual, onde os recursos já estão todos disponibilizados. No entanto, em alguns casos a segunda execução apresentou tempo de execução superior à primeira execução, especialmente quando o coletor de lixo da máquina virtual precisou liberar espaço para a instanciação de novos objetos.

O servidor usado nos experimentos possui processador Intel Core i7 2.93GHz e 16 GB de memória RAM DDR3 1333 MHz. Nele foi instalada a aplicação OpenLink Virtuoso OpenSource para armazenamento dos dados RDF e para prover o serviço de Endpoint SPARQL. A máquina cliente usada durante os experimentos possui processador Intel Core 2 Duo 2.93GHz e 2GB de memória RAM 667 MHz. Ela foi responsável pela execução das consultas federadas a partir das ferramentas Jena, Sesame, FedX e QEF. Antes de adotarmos o Virtuoso como Endpoint SPARQL e RDF Store, utilizamos o Fuseki do projeto Apache Jena. No entanto, algumas das consultas chegavam a sobrecarregar tanto o servidor, que ele não suportava a carga de trabalho e interrompia o serviço. Isso mostra que há ainda muitas melhorias a serem realizas

sobre os Endpoints existentes.

Durante os experimentos, tanto o servidor quanto a máquina cliente ficaram dedicados a essas tarefas em um ambiente controlado, onde a conexão entre ambos ocorreu através de uma rede local. Nesse contexto, a intenção foi evitar que interferências de outras atividades viessem a adulterar os resultados dos experimentos. Além disso, somente consideramos os resultados semelhantes ocorridos durante os 10 ciclos de execução. Resultados considerados como valores atípicos (*outliers*) em relação aos demais valores obtidos foram descartados. Felizmente esses resultados praticamente não ocorreram justamente devido à adoção de um ambiente controlado.

Os dados RDF foram importados para a RDF Store a partir de dumps de conjuntos de dados disponíveis na Web sob licença aberta. Os dados de *diseasome, dailymed, sider, drugbank* e *dblp* foram importados por completo. Os dados importados de *linkedgeodata* incluíram somente as triplas com ligações para *dbpedia*. Já os dados usados da *DBpedia* foram apenas aqueles relacionados a coordenadas geográficas.

Por várias vezes tentamos também realizar os mesmos experimentos sobre os dados originais disponíveis na Web. No entanto, as consultas sobrecarregavam os endpoints utilizados, chegando a causar a interrupção do serviço em certos casos. Esse problema ocorreu principalmente quando consultamos *Endpoints* que adotavam o servidor D2R. Dos sete conjuntos de dados que usamos, cinco deles (*diseasome, dailymed, sider, drugbank* e *dblp*) usavam o *D2R Server* como *Endpoint SPARQL*. Em outros casos, o servidor limitava os resultados inserindo um fim de arquivo antecipado que ocasionava uma exceção com a mensagem *"Premature end of file"* antes mesmo de chegar aos quatro minutos de execução da consulta. Infelizmente esses problemas dificultaram os experimentos no ambiente não controlado e real da Web que contém os conjuntos de dados originais que usamos. Mesmo assim, pretendemos como trabalho futuro realizar esses experimentos em ambiente Web com uma seletividade maior a fim de viabilizar a obtenção dos resultados.

A seguir serão apresentadas descrições e listagens das consultas usadas nos experimentos. As listagens dos *templates* do QEF com planos de consulta equivalentes às consultas que serão descritas a seguir podem ser encontradas no Apêndice A.

Consultas para avaliação das estratégias de junção

Foram criadas três consultas para permitir a comparação entre as diferentes estratégias de junção. As consultas Q1 e Q2 diferem principalmente pela quantidade de resultados obtidos. Enquanto a primeira retorna 43016 resultados, a segunda retorna apenas 6124. Essa diferenciação afeta os resultados obtidos, conforme será visto adiante. Já a consulta Q3 obtém uma grande quantidade de resultados (86516), mas difere das demais por fazer uso de duas junções e não apenas de uma junção, como ocorre em Q1 e Q2.

A consulta Q1 (Figura 7.1) obtém URIs de recursos do conjunto de dados linkedgeodata, juntamente com suas respectivas latitudes e longitudes obtidos a partir do conjunto de dados DBpedia. Ao todo são obtidos 43016 resultados a partir da execução dessa consulta.

A consulta Q2 (Figura 7.2) obtém URIs de doenças e de possíveis drogas usadas

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX geopos: <http://www.w3.org/2003/01/geo/wgs84_pos#>

SELECT ?s ?lat ?long
WHERE {
  SERVICE <http://linkedgeodata.arida.ufc.br/sparql>  {
    ?s owl:sameAs ?geo .
  }
  SERVICE <http://dbpedia.arida.ufc.br/sparql> {
    ?geo geopos:lat ?lat ;
         geopos:long ?long .
  }
}
```

Figura 7.1: Consulta SPARQL Federada Q1 para avaliação das estratégias de junção

para tratar cada doença a partir da fonte de dados diseasome. Além desses dados, os nomes completos das drogas usadas no tratamento da doença são obtidos na fonte de dados *dailymed*. Um total de 6124 resultados são obtidos após a execução de Q2.

```
PREFIX diseasome: <http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/>
PREFIX dailymed: <http://www4.wiwiss.fu-berlin.de/dailymed/resource/dailymed/>

SELECT DISTINCT ?ds ?dg ?dgn
WHERE {
  SERVICE <http://diseasome.arida.ufc.br/sparql> {
    ?ds diseasome:possibleDrug ?dg .
  }
  SERVICE <http://dailymed.arida.ufc.br/sparql> {
    ?dg dailymed:fullName ?dgn .
  }
}
```

Figura 7.2: Consulta SPARQL Federada Q2 para avaliação das estratégias de junção

A consulta Q3 (Figura 7.3) obtém nomes de princípios ativos de drogas no conjunto de dados *dailymed*. Depois disso, são verificadas as ligações *owl:sameAs* com *sider* e ligações *dailymed:genericDrug* com *drugbank* para obtenção de nomes dos efeitos colaterais da droga no *sider*, e de sua fórmula química no *drugbank*, totalizando 86516 resultados.

Consultas para avaliação das estratégias de junção à esquerda

As consultas Q4, Q5 e Q6 são semelhantes às consultas Q1, Q2 e Q3, respectivamente. No entanto, as primeiras possuem o operador de junção à esquerda (*left join*), enquanto que as últimas usam o operador de junção. Desse modo, os valores obtidos do lado esquerdo de cada operador da junção à esquerda são usados para construção do resultado final, mesmo que não haja correspondência com os resultados obtidos do lado direito. Uma consequência imediata disso é a obtenção de resultados adicionais na junção à esquerda, quanto comparada com a junção.

A consulta Q4 (Figura 7.4) obtém URIs de recursos com possíveis correspondentes na DBpedia. Caso a correspondência realmente exista na DBpedia, sua latitude e longitude serão preenchidas no resultado. Em caso contrário, latitude e longitude aparecerão no resultado com valores nulos. A consulta Q4 retorna um total de 103631 resultados.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dailymed: <http://www4.wiwiss.fu-berlin.de/dailymed/resource/dailymed/>
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX sider: <http://www4.wiwiss.fu-berlin.de/sider/resource/sider/>

SELECT ?dgain ?dgcf ?sen
WHERE {
  SERVICE <http://dailymed.arida.ufc.br/sparql> {
    ?dg dailymed:activeIngredient ?dgai .
    ?dgai rdfs:label ?dgain .
    ?dg dailymed:genericDrug ?gdg .
    ?dg owl:sameAs ?sa .
  }
  SERVICE <http://sider.arida.ufc.br/sparql> {
    ?sa sider:sideEffect ?se .
    ?se sider:sideEffectName ?sen .
  }
  SERVICE <http://drugbank.arida.ufc.br/sparql> {
    ?gdg drugbank:chemicalFormula  ?dgcf .
  }
}
```

Figura 7.3: Consulta SPARQL Federada Q3 para avaliação das estratégias de junção

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX geopos: <http://www.w3.org/2003/01/geo/wgs84_pos#>

SELECT ?s ?lat ?long
WHERE {
  SERVICE <http://linkedgeodata.arida.ufc.br/sparql>  {
    ?s owl:sameAs ?geo .
  }
  OPTIONAL {
    SERVICE <http://dbpedia.arida.ufc.br/sparql> {
      ?geo geopos:lat ?lat ;
           geopos:long ?long .
    }
  }
}
```

Figura 7.4: Consulta SPARQL Federada Q4 para avaliação das estratégias de junção à esquerda

A consulta Q5 (Figura 7.5) obtém URIs de doenças com possíveis drogas usadas em seu tratamento. Caso essa droga esteja presente no conjunto de dados dailymed, o valor do seu nome completo será preenchido no resultado final, ou ficará nulo, em caso contrário. Ao todo essa consulta retorna 14325 resultados.

A consulta Q6 (Figura 7.6) obtém 99222 resultados com os nomes dos princípios ativos de drogas do conjunto de dados dailymed. A partir disso, os predicados owl:sameAs e dailymed:genericDrug são usados para designar recursos que representam a mesma entidade nos conjuntos de dados sider e drugbank, respectivamente. Correspondências não satisfeitas no sider ou drugbank, resultarão em valores nulos de fórmula química ou de nomes de efeitos colaterais.

Consultas para avaliação das estratégias de união

A duas consultas criadas para permitir uma comparação de desempenho das estratégias de união diferem na quantidade de uniões realizadas. Enquanto a consulta Q7 tem uma única operação

```
PREFIX diseasome: <http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/>
PREFIX dailymed: <http://www4.wiwiss.fu-berlin.de/dailymed/resource/dailymed/>
SELECT DISTINCT ?ds ?dg ?dgn
WHERE {
  SERVICE <http://diseasome.arida.ufc.br/sparql>  {
     ?ds diseasome:possibleDrug ?dg .
  }
  OPTIONAL {
    SERVICE <http://dailymed.arida.ufc.br/sparql>  {
        ?dg dailymed:fullName ?dgn .
    }
  }
}
```

Figura 7.5: Consulta SPARQL Federada Q5 para avaliação das estratégias de junção à esquerda

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dailymed: <http://www4.wiwiss.fu-berlin.de/dailymed/resource/dailymed/>
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX sider: <http://www4.wiwiss.fu-berlin.de/sider/resource/sider/>

SELECT ?dgain ?dgcf ?sen
WHERE {
  SERVICE <http://dailymed.arida.ufc.br/sparql> {
    ?dg dailymed:activeIngredient ?dgai .
    ?dgai rdfs:label ?dgain .
    ?dg dailymed:genericDrug ?gdg .
    ?dg owl:sameAs ?sa .
  }
  OPTIONAL {
    SERVICE <http://sider.arida.ufc.br/sparql> {
      ?sa sider:sideEffect ?se .
      ?se sider:sideEffectName ?sen .
    }
  }
  OPTIONAL {
    SERVICE <http://drugbank.arida.ufc.br/sparql> {
      ?gdg drugbank:chemicalFormula  ?dgcf .
    }
  }
}
```

Figura 7.6: Consulta SPARQL Federada Q6 para avaliação das estratégias de junção à esquerda

de União, a consulta Q8 possui dez operadores de União.

A consulta Q7 (Figura 7.7) realiza a união de nomes genéricos e indicações de drogas entre os conjuntos de dados drugbank e dailymed, totalizando 5146 resultados.

A consulta Q8 (Figura 7.8) faz a união de nomes de pesquisadores e suas publicações em conjuntos de dados do dblp e obtém 18327 resultados no total. Cada conjunto de dados seria requisitado por nomes de pesquisadores contendo "ab"como segundo e terceiro caracteres, e tendo o primeiro caractere diferenciado uns dos outros e com variação de "A"a "J".

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX dailymed: <http://www4.wiwiss.fu-berlin.de/dailymed/resource/dailymed/>

SELECT ?gn ?indication
WHERE {
  {
    SERVICE <http://drugbank.arida.ufc.br/sparql> {
      ?dn drugbank:genericName ?gn ;
          drugbank:indication ?indication.
    }
  }
  UNION {
    SERVICE <http://dailymed.arida.ufc.br/sparql> {
       ?dn dailymed:name ?gn ;
           dailymed:indication ?indication .
    }
  }
}
```

Figura 7.7: Consulta SPARQL Federada Q7 para avaliação das estratégias de união

```
PREFIX dc:   <http://purl.org/dc/elements/1.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?label ?pub_title where {
  {
    SERVICE <http://dblp01.arida.ufc.br/sparql> {
         ?publication dc:creator ?dblp_researcher ;
                      dc:title ?pub_title .
         ?dblp_researcher rdfs:label ?label .
         FILTER regex(?label, "^Aab")
    }
  } UNION {
    SERVICE <http://dblp02.arida.ufc.br/sparql> {
        ?publication dc:creator ?dblp_researcher ;
                     dc:title ?pub_title .
        ?dblp_researcher rdfs:label ?label .
        FILTER regex(?label, "^Bab")
    }

  ...

  } UNION {
    SERVICE <http://dblp09.arida.ufc.br/sparql> {
        ?publication dc:creator ?dblp_researcher ;
                     dc:title ?pub_title .
        ?dblp_researcher rdfs:label ?label .
        FILTER regex(?label, "^Iab")
    }
  } UNION {
    SERVICE <http://dblp10.arida.ufc.br/sparql> {
        ?publication dc:creator ?dblp_researcher ;
                     dc:title ?pub_title .
        ?dblp_researcher rdfs:label ?label .
        FILTER regex(?label, "^Jab")
    }
  }
}
```

Figura 7.8: Consulta SPARQL Federada Q8 para avaliação das estratégias de união

7.3 Análise dos Resultados

Esta seção analisa os resultados dos experimentos relacionados a consultas contendo junções, junções à esquerda e uniões nas diferentes estratégias de execução adotadas. As principais informações coletadas para estabelecer comparações entre o desempenhos das consultas foram o tempo de execução da consulta e a quantidade máxima de memória consumida pela máquina virtual Java durante um ciclo de execução, que é constituído de duas execuções da mesma consulta (Execução 1 e Execução 2).

7.3.1 Avaliação de experimentos relacionados às consultas com uso da operação de junção

As ferramentas Jena, Sesame e FedX foram usadas para executar as consultas federadas Q1, Q2 e Q3. O QEF foi usado para executar planos de consulta correspondentes às consultas citadas. Basicamente no QEF foram criados três planos de consulta relativos a cada uma das consultas federadas (Q1, Q2, e Q3). Esses planos escritos como *templates* do QEF estão listados no Apêndice A. No primeiro plano de consulta o operador de junção usado foi o *BindJoin* (BJ) apresentado na Seção 6.4. O operador *BindJoin* é equivalente ao uso do operador *SetBindJoin* sem o uso de threads e com set de tamanho um. Já os operadores usados nos demais planos de consulta foi o *SetBindJoin* (SBJ) tratado na Seção 6.5. A distinção entre o segundo e o terceiro planos ocorreu em relação ao não uso (SBJ) e ao uso (SBJ-T), respectivamente, de paralelização na execução das consultas através do uso de *threads*. Resumindo: o primeiro plano criado para o QEF usa o operador *BindJoin* (BJ), o segundo plano usa o operador *SetBindJoin* sem a utilização de threads (SBJ) e, finalmente, o terceiro plano faz uso do operador *SetBindJoin* com o uso de threads (SBJ-T). O valor escolhido como número máximo de threads simultâneas usado em todas as consultas que envolveram o operador SetBindJoin foi o número cem. Sua seleção deveu-se ao fato de que ao fixarmos os demais parâmetros do SetBindJoin e variarmos somente esse número, obtivemos o melhor desempenho quando ele estava próximo de cem. Sobre a quantidade máxima de threads simultâneas, observamos que quando não definíamos essa limitação, o número excessivo de threads simultâneas ocasionava um grande número de sockets abertos com conexões em espera. Isso terminava gerando uma exceção que interrompia a execução da consulta (`java.net.SocketException: Too many open files`). Por outro lado, poucas threads simultâneas pode resultar em uma menor vazão (throughput), desde que mais threads poderiam estar recebendo resultados. A conclusão a que chegamos, é que há valores para o número máximo de threads simultâneas que se aproximam de um ponto de equilíbrio entre produção (Endpoint SPARQL) e consumo (QEF-LD) de resultados, que maximiza a vazão (throughput). Para o ambiente dos nossos experimentos, esse valor era próximo do número cem.

A consulta Q1 foi executada através de todas as ferramentas já mencionadas. No entanto, não foi possível obter resultados através do Sesame em sua versão 2.6.5. A consulta é executada, a aplicação não trava, mas também não retorna resultados mesmo após horas de execução. Nenhum resultado é obtido, nem ocorre uma exceção indicativa do problema. Foi constatado também que não há um consumo excessivo de memória. A partir do segundo minuto

de execução a máquina virtual Java passa a usar uma quantidade fixa de memória, estabilizando em um único valor. Diferentes ciclos de execução consumiram as seguintes quantidades de memória RAM: 159,75MB, 163,06MB e 180MB. O problema pode estar relacionado a algum bug no suporte a consultas federadas do Sesame. Vale ressaltar que esse suporte somente foi disponibilizado no Sesame a partir de sua versão 2.6.0 lançada em Outubro de 2011.

Também não foi possível obter todos os resultados através do FedX na máquina cliente que dispunha de 2GB de memória RAM. Sempre que requisitada a execução da consulta Q1, o FedX consumia toda a memória disponível para a máquina virtual e, depois de algum tempo, lançava uma exceção relacionada à falta de memória disponível (*OutOfMemoryException*). Ao executar a mesma consulta no próprio servidor com 16GB de memória RAM, a máquina virtual Java chegou a consumir 8031MB para concluir satisfatoriamente a execução da consulta. Isso indica um excessivo consumo de memória pelo mediador FedX ao processar junções com elevado número de resultados, como é o caso da consulta Q1 que deveria obter 43016 resultados.

A Figura 7.9 demonstra que a estratégia mais eficiente em termos de tempo de execução para da consulta Q1 foi o SetBindJoin com o uso de threads e usando um set de tamanho 20 implementado no QEF, seguido do SetBindJoin sem o uso de threads e com um set de tamanho 25. A primeira execução de todas as estratégias revelou um tempo ligeiramente maior que o da segunda execução, devido à alocação de recursos pela máquina virtual Java. No entanto, esse tempo adicional revelou-se inexpressivo em relação ao tempo total de execução da consulta.

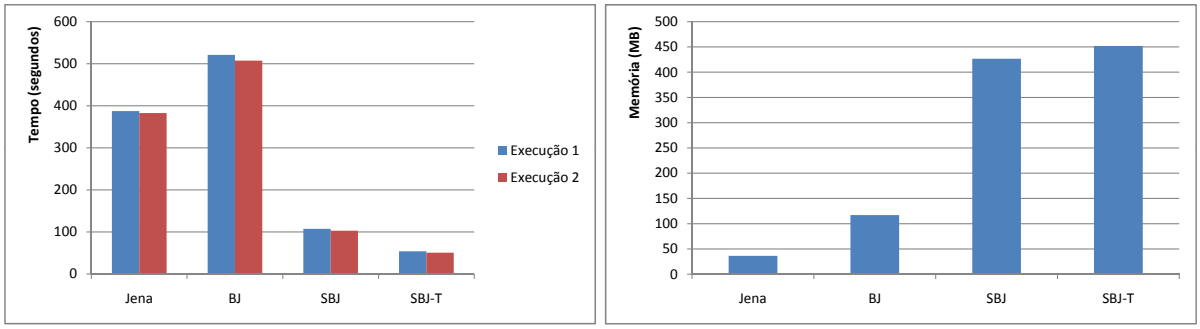

Figura 7.9: Gráfico comparativo de tempo de execução e uso de memória dos resultados da consulta Q1

Por outro lado, o *SetBindJoin* com o uso de *threads* foi a estratégia que mais consumiu memória, dentre as que obtiveram todos os resultados. Esse consumo deveu-se principalmente à necessidade de armazenamento dos dados necessários à construção dos resultados das junções sobre os sets. O uso de threads eleva um pouco mais esse consumo de memória, devido ao processamento de mais itens de forma simultânea. O Jena revelou-se mais eficiente no consumo de memória.

Através dos resultados obtidos também é possível observar que é possível melhorar o desempenho da execução do QEF, tendo em vista que o *BindJoin* obteve resultados inferiores ao Jena nos itens tempo de execução e uso de memória, embora usando o mesmo algoritmo de junção (*BindJoin*).

A análise da Figura 7.10 mostra que a diferença de desempenho entre o *SetBindJoin* sem o uso de threads e com o uso de threads diminui à medida que o tamanho máximo dos sets aumenta e atinge uma diferença relativamente estável a partir de um determinado valor para o tamanho máximo de sets. Observa-se também que a partir de um determinado valor, aumentar o tamanho dos sets torna as consultas mais lentas. Portanto, é importante encontrar um ponto de equilíbrio entre a produção e o consumo dos resultados para maximizar o desempenho das consultas. Nos experimentos realizados sets com tamanhos máximos entre 20 e 25 obtiveram os menores tempos de execução.

O tamanho máximo dos sets usado nas consultas Q1 e Q3 foi 57. Não foi possível usar sets de tamanhos maiores porque o servidor Virtuoso limita uma consulta a possuir no máximo 56 operações de União. Como a estratégia de *BINDING* usado pelo *SetBindJoin* envolve a reformulação da consulta através da criação de várias operações de União (Seção 6.5), não foi possível criar *sets* maiores para essas consultas.

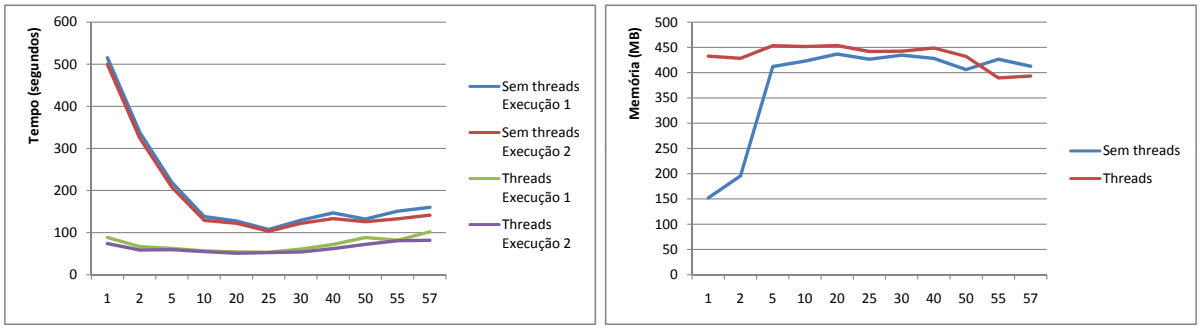

Figura 7.10: Gráfico comparativo de tempo de execução e uso de memória dos resultados da consulta Q1 usando o algoritmo SetBindJoin para diferentes tamanhos de set.

Os menores tempos de execução da consulta Q2 foram obtidos através da execução do SetBindJoin com o uso de threads, seguido pelo mesmo operador sem o uso de threads, conforme pode ser observado na Figura 7.11. Esses tempos foram bem menores e, portanto, com melhores desempenhos que os demais. A seguir vem o FedX com desempenho bastante inferior ao SetBindJoin, mas ainda melhor que as outras estratégias.

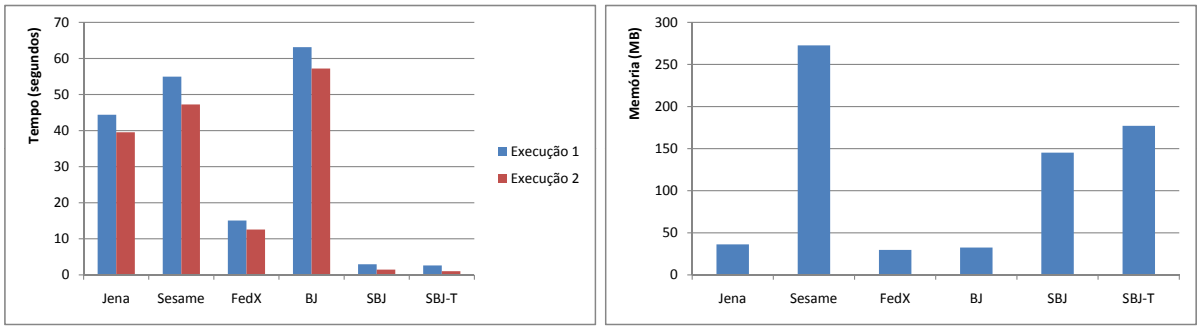

Figura 7.11: Gráfico comparativo de tempo de execução e uso de memória dos resultados da consulta Q2.

A Figura 7.12 revela que para pequenos tamanhos de sets o SetBindJoin com o uso de threads é executado em tempo muito inferior que o SetBindJoin sem o uso de threads.

No entanto, essa diferença diminui à medida que o tamanho dos sets aumenta. Em relação à memória, ocorre o inverso, ou seja, a estratégia com o uso de threads consome muito mais memória que o SetBindJoin sem threads para sets com poucos elementos. Não há praticamente diferença entre o consumo de memória com ou sem o uso de threads, quando os sets possuem uma quantidade de elementos igual ou superior a 50.

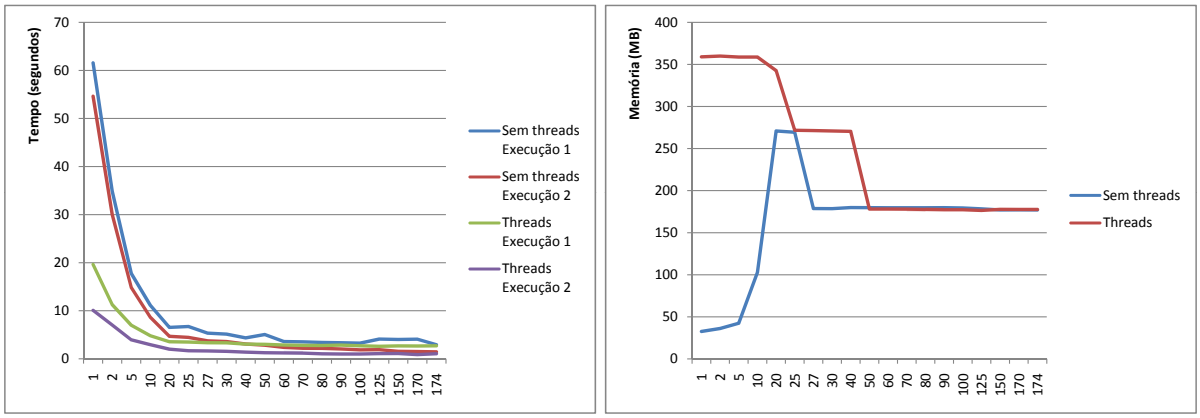

Figura 7.12: Gráfico comparativo de tempo de execução e uso de memória dos resultados da consulta Q2 usando o algoritmo SetBindJoin para diferentes tamanhos de set.

Na consulta Q3, assim como na consulta Q1, também não foi possível obter os resultados do FedX devido ao elevado consumo de memória que novamente acabou ocasionando uma exceção por falta de memória disponível na máquina virtual Java. A consulta Q3 distingue-se das demais consultas com junções, por realizar duas operações de junção, ao invés de apenas uma. Q3 é também a consulta com junção que obtém a maior quantidade de resultados. São obtidos 86516 resultados no total. Essa consulta chegou a causar interrupção nos serviços dos *Endpoints* remotos disponíveis na Web de Dados dos conjuntos de dados *Sider*, *DailyMed* e *DrugBank*. Estabelecendo um comparativo com relação ao uso de memória entre Q1 e Q3, percebe-se que o uso de memória em Q3 foi muito maior que em Q1 nas estratégias Jena e BJ. No entanto, SBJ e SBJ-T não tiveram aumento muito significativo no consumo de memória. Sesame revelou-se bastante modesto no consumo de memória, obtendo o melhor resultado nesse quesito. Em relação ao tempo de execução, SBJ e SBJ-T novamente obtiveram um desempenho bastante superior às demais estratégias, como pode ser observado na Figura 7.13.

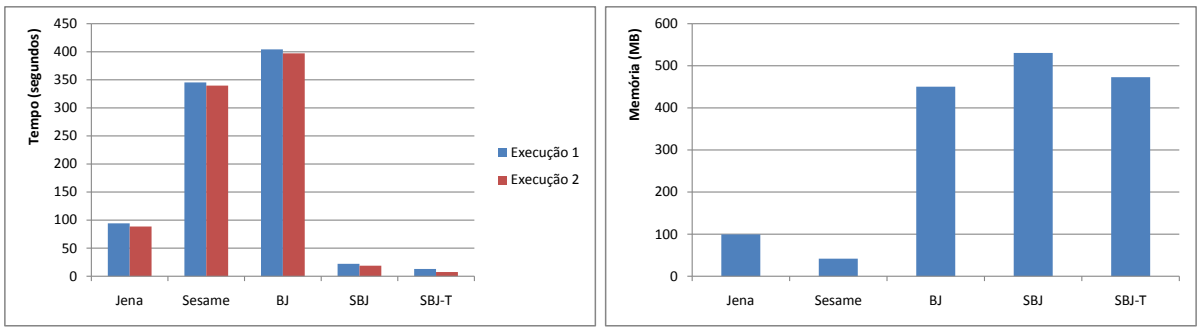

Figura 7.13: Gráfico comparativo de tempo de execução e uso de memória dos resultados da consulta Q3

A Figura 7.14 revela um uso de memória semelhante entre o uso do SBJ com ou

sem o uso de threads, sendo a estratégia com uso de threads ligeiramente melhor nesse aspecto. Em relação ao tempo de execução, o SBJ com uso de threads possui desempenho muito melhor para sets com tamanhos reduzidos, no entanto à medida que sets maiores são usados, essa diferença de desempenho diminui, até se tornar muito pequena para sets com cerca de 30 ou mais elementos.

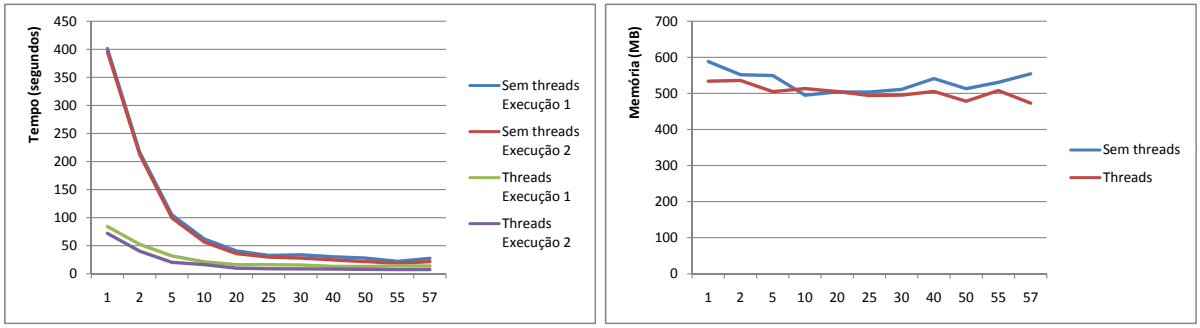

Figura 7.14: Gráfico comparativo de tempo de execução e uso de memória dos resultados da consulta Q3 usando o algoritmo SetBindJoin para diferentes tamanhos de set

7.3.2 Avaliação de experimentos relacionados às consultas com uso da operação de junção à esquerda

Os experimentos realizados usando as consultas Q4, Q5 e Q6 revelam o Jena com tempos de execução inferiores às demais estratégias e o Sesame com resultados próximos, mas com tempos de execução ligeiramente maiores, conforme pode ser observado nas Figuras 7.15, 7.16 e 7.17. Por outro lado, o Sesame apresentou consumo de memória muito inferior ao Jena especialmente nas consultas Q4 e Q6 (Figuras 7.15 e 7.17).

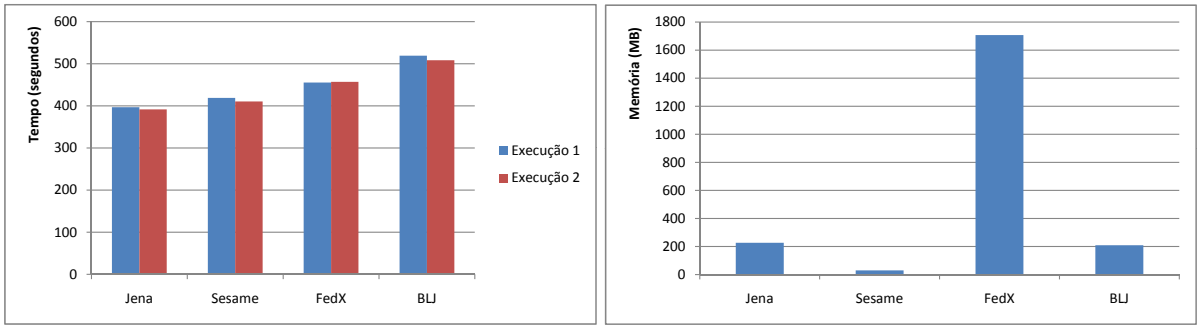

Figura 7.15: Gráfico comparativo de tempo de execução e uso de memória dos resultados da consulta Q4

O FedX revelou-se um grande consumidor de memória em todas as consultas com junção à esquerda, chegando a consumir em certos casos toda a memória disponibilizada pela máquina virtual Java. O operador BindLeftJoin implementado no QEF com poucas alterações com relação ao BindJoin, também apresentou desempenhos inferiores às demais estratégias na maioria das consultas. Isso significa que é preciso melhorar o desempenho do QEF para execução desses operadores simples.

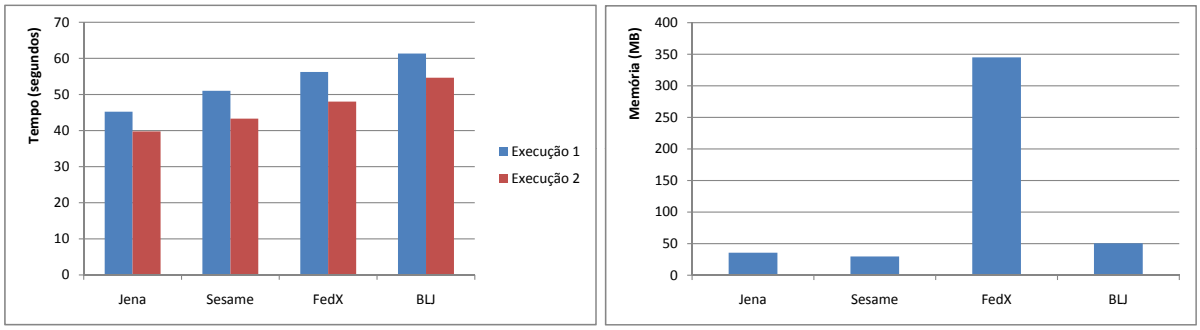

Figura 7.16: Gráfico comparativo de tempo de execução e uso de memória dos resultados da consulta Q5

Um resultado fora do padrão ocorreu na Execução 2 do FedX na consulta Q6 que pode ser observado na Figura 7.17. Nesse caso, o FedX obteve um tempo de sua segunda execução muito superior ao tempo de sua primeira execução e também em relação aos tempos das demais estratégias. Isso ocorreu devido ao consumo quase que integral de toda a memória disponível na máquina virtual Java, o que levou a um trabalho excessivo para liberação de espaço, efetuado pelo coletor de lixo da JVM, especialmente durante a segunda execução.

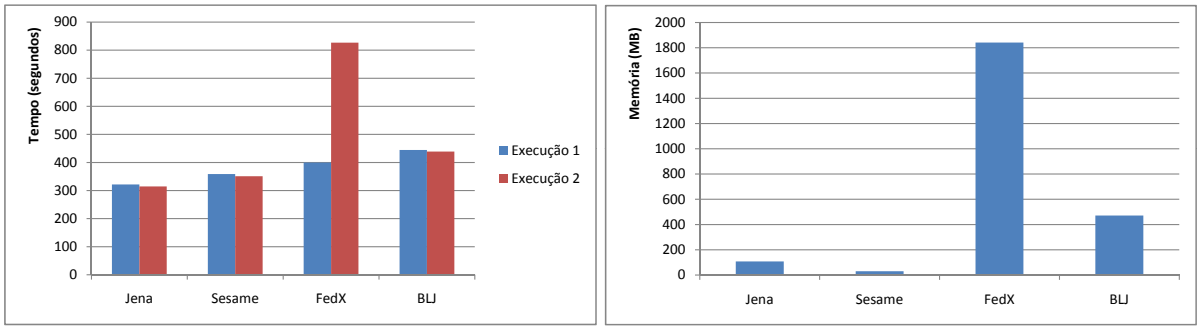

Figura 7.17: Gráfico comparativo de tempo de execução e uso de memória dos resultados da consulta Q6

7.3.3 Avaliação de experimentos relacionados às consultas com uso da operação de união

Em relação ao uso da operação de união, os resultados de modo geral foram mais uniformes entre as diversas estratégias usadas. Sesame e FedX destacaram-se pelo menor consumo de memória em relação às demais ferramentas, obtendo resultados semelhantes neste aspecto. Pela primeira vez nos experimentos realizados, o FedX obteve resultados satisfatórios com relação ao uso de memória.

A Figura 7.18 mostra que o tempo de execução da consulta Q7 foi bastante curto, impactando na diferença entre o tempo da primeira e a segunda execuções. Isso se deve ao fato de que durante a primeira execução ainda há bastante trabalho realizado pela máquina virtual Java para alocação de recursos. Todas as estratégias obtiveram resultados semelhantes com relação ao tempo de execução, sendo que o Jena demorou mais que os demais para obter os resultados, especialmente na primeira execução da consulta. A estratégia Union-T implementada

no QEF obteve um tempo de execução ligeiramente melhor que os demais durante a segunda execução. Sobre memória, Sesame e FedX consumiram pelo menos da metade da memória usada pelas outras ferramentas.

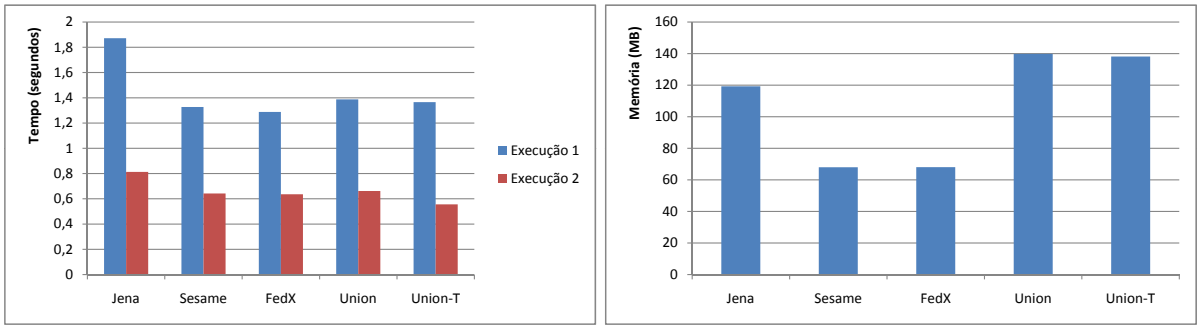

Figura 7.18: Gráfico comparativo de tempo de execução e uso de memória dos resultados da consulta Q7

A Figura 7.19 apresenta resultados praticamente idênticos entre os tempos de execução de Jena, Sesame e Union sem uso de threads implementado no QEF. No entanto, ao fazerem uso de threads em seus algoritmos, FedX e Union-T alcançam tempos de execução semelhantes e quase duas vezes mais rápidos que as demais estratégias. O consumo de memória de Union e Union-T ficaram muito próximos entre si, mas maior em comparação com as demais estratégias, o que significa que se deve trabalhar na redução do consumo de memória realizado por essas abordagens.

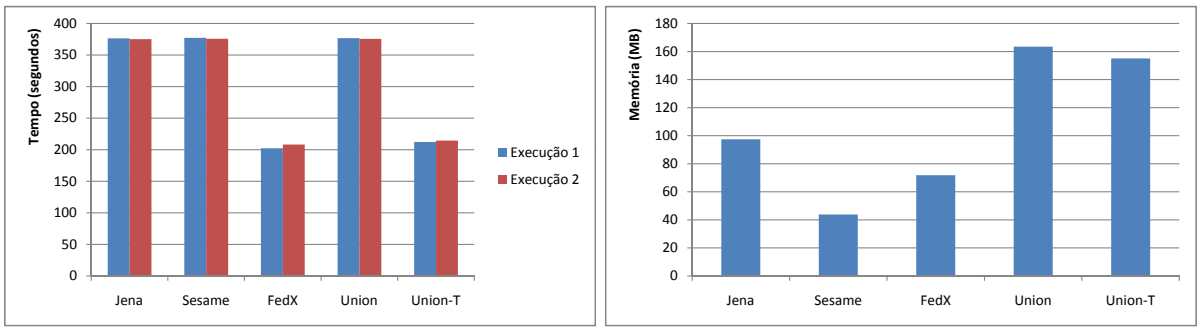

Figura 7.19: Gráfico comparativo de tempo de execução e uso de memória dos resultados da consulta Q8

7.4 Conclusões

Consultas em ambiente controlado usando o operador SetBindJoin implementado no QEF obteve tempos de execução consideravelmente menores que as outras estratégias usadas nos experimentos, mesmo para um grande volume de dados processados. Os melhores resultados foram obtidos com ou sem o uso threads para sets com maior quantidade de elementos, causando uma redução no número de chamadas a Endpoints remotos e, consequentemente, reduzindo o tempo de execução. Assim, minimizar o número de requisições remotas intermediárias para construção do resultado final, leva a melhorias bastante significativas no desempenho

das consultas. Por outro lado, o acúmulo de resultados em sets causa um aumento do consumo de memória. Felizmente esse aumento não se torna excessivo, desde que o espaço de memória reservado aos sets pode ser liberado logo após o seu processamento.

Os resultados relacionados ao operador Union também foram satisfatórios. Union sem o uso de threads obteve tempos de execução semelhantes ao Jena e Sesame que também não usam threads. As estratégias de união com uso de threads, que no caso foram Union implementado no QEF e FedX, alcançaram tempos de execução muito menores e, portanto com melhor desempenho, que as demais estratégias para uma grande quantidade de resultados e uso de vários operadores de união em uma mesma consulta. Isso demonstra que o aumento do grau de paralelismo permite que mais conexões simultâneas estejam transferindo dados, aumentando, assim, a vazão (throughput).

Por fim, os operadores BindJoin e BindLeftJoin, cujos algoritmos são simples, muito similares e não buscam reduzir o número de conexões remotas, apresentaram tempos de execução superiores às demais abordagens analisadas, necessitando, portanto, de aperfeiçoamentos. Com relação, à operação de junção à esquerda, pode-se criar um operador semelhante ao SetBindJoin para aliviar o número de chamadas remotas e melhorar bastante o tempo de execução.

8 CONCLUSÃO

8.1 Considerações Finais

Este trabalho apresentou um módulo (QEF-LD) para o processamento eficiente de consultas federadas sobre Linked Data, que pode ser usado em arquiteturas com ou sem o uso de um mediador. O módulo estendeu o *QEF – Query Evaluation Framework* para execução dos planos de consulta federados sobre Linked Data. Para isso foram implementados operadores da álgebra SPARQL, além de um *datasource* específico para acesso à Linked Data.

O trabalho ainda propôs arquiteturas de Linked Data Mashups com e sem uso de mediador, que possuem o QEF-LD como um de seus componentes. A arquitetura sem o uso de mediador baseia-se no conceito de *Linked Data Mashup Services (LIDMS)*, que são serviços Web REST capazes de recuperar dados a partir do recebimento de uma URI. Cada URI possui uma correspondência com um plano de consulta federado e possivelmente parametrizado, gerado em tempo de projeto. A especificação e implementação do ambiente para execução de LIMDS (LEXEN) são importantes contribuições deste trabalho.

Por fim, foram realizados experimentos para validar a viabilidade de uso do QEF-LD no processamento de planos de consulta federados sobre a web de dados, a partir de comparações com outras estratégias existentes e voltadas para integração de dados sobre Linked Data.

8.2 Trabalhos Futuros

Muitas ideias surgiram durante o desenvolvimento deste trabalho. Algumas foram deixadas de lado por não terem se mostrado viáveis. Outras foram postas em prática e fazem parte desta dissertação. No entanto, o tempo que tivemos mostrou-se ainda curto para a execução de muitas outras atividades que vamos aqui delinear como possíveis trabalhos futuros. Elas foram agrupadas a seguir, de acordo com principais assuntos apresentados na dissertação.

Processador de Consultas Federadas (QEF-LD)

- Realizar experimentos no ambiente não controlado e real da Web. Esse tipo de experimento será ainda mais necessário quando operadores adaptativos forem implementados.
- Melhorar o desempenho das execuções dos operadores da álgebra SPARQL já implementados no QEF, tanto na diminuição do tempo de resposta, quanto em relação ao consumo de memória.
- Implementar operações da álgebra SPARQL ainda não implementadas no QEF. Com relação a operação de junção à esquerda, um avanço seria a implementação de um operador semelhante ao SetBindJoin já implementado, para aliviar o número de chamadas remotas e melhorar bastante o tempo de execução (ver Seção 7.4).

- Implementar estratégias de adaptatividade aos operadores da álgebra SPARQL implementados no QEF.
- Implementar as formas de consulta CONSTRUCT, DESCRIBE e ASK, desde que a implementação atual do QEF-LD somente lida com a forma de consulta SELECT da linguagem SPARQL.
- Usar cache de dados e índices para melhorar o desempenho das consultas.

Ambiente de Execução de LIMDS

- Adicionar suporte ao Ambiente de Execução de LIDMS (LEXEN) para formato de saída texto plano, bem como para CSV e TSV.
- Usar *RDF Store* para armazenamento de metadados e visões materializadas com a finalidade de reduzir o tempo de resposta das consultas. A definição das visões materializadas ocorre em tempo de projeto, a partir de indicadores do que deve ser materializado. Os filtros usados nas consultas seriam um desses indicadores. Políticas específicas podem ser usadas para invalidação e recarga dos dados materializados. As estratégias atuais de acesso à Web de dados normalmente endereçam apenas o acesso virtual ou apenas materializado dos dados provenientes de *Linked Data*. Uma abordagem híbrida que busca aproveitar vantagens de ambas seria um diferencial em relação aos trabalhos existentes.

Ferramenta para construção de LIMDS

- Implementar uma ferramenta para permitir a especificação e construção de LIDMS, inclusive com a geração dos planos de consulta federados a partir de consultas SPARQL sobre a *OD*. Essa ferramenta ainda deve permitir que consultas sejam testadas e avaliadas ainda na fase de projeto, para que o desenvolvedor possa ter um *feedback* mais instantâneo sobre uma consulta criada ou editada.
- Incluir uma fase de pós-processamento ao processo de geração dos planos de consulta federados. Essa etapa tem como finalidade a realização automática de operações para remover conflitos, inconsistências e duplicação de resultados das consultas.

Mediador

- Implementar um mediador de acordo com a arquitetura especificada no Capítulo 4.

Outros

- Estabelecer de forma automática, *links* virtuais entre ontologias ou instâncias.

Percebe-se, assim, que há vários caminhos a percorrer para estender e aperfeiçoar o trabalho apresentado. Esperamos que eles possam ser trilhados para que mais contribuições relevantes possam advir.

REFERÊNCIAS BIBLIOGRÁFICAS

ALEXANDER, K. et al. Describing Linked Datasets - On the Design and Usage of voiD, the 'Vocabulary of Interlinked Datasets'. In: *WWW 2009 Workshop: Linked Data on the Web (LDOW2009)*. Madrid, Spain: [s.n.], 2009.

ALEXANDER, K. et al. *Describing Linked Datasets with the VoID Vocabulary*. 2011. `http://www.w3.org/TR/void/`.

ARAÚJO, S. F. C.; SCHWABE, D. Explorator: a Tool for Exploring RDF Data Through Direct Manipulation. In: *LDOW 2009: Linked Data on the Web*. [S.l.: s.n.], 2009.

ARAÚJO, S. F. C.; SCHWABE, D.; BARBOSA, S. D. J. Experimenting with Explorator: a Direct Manipulation Generic RDF Browser and Querying Tool. In: *Workshop on Visual Interfaces to the Social and the Semantic Web (VISSW2009)*. [S.l.: s.n.], 2009.

AUER, S. et al. Triplify: Light-weight linked data publication from relational databases. In: QUEMADA, J. et al. (Ed.). *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*. [S.l.]: ACM, 2009. p. 621–630. ISBN 978-1-60558-487-4.

BECKER, C.; BIZER, C. DBpedia Mobile: A Location-Enabled Linked Data Browser. In: *Linked Data on the Web (LDOW2008)*. [S.l.: s.n.], 2008.

BECKETT, D.; BROEKSTRA, J. *SPARQL Query Results XML Format*. 2008. `http://www.w3.org/TR/rdf-sparql-XMLres/`.

BERNERS-LEE, T. *Linked Data - Design Issues*. 2006. `http://www.w3.org/DesignIssues/LinkedData.html`.

BERNERS-LEE, T. et al. Tabulator: Exploring and Analyzing Linked Data on the Semantic Web. In: *In Procedings of the 3rd International Semantic Web User Interaction Workshop (SWUI06*. [S.l.: s.n.], 2006. p. 06.

BERNERS-LEE, T.; FIELDING, R.; MASINTER, L. *RFC 3986 – Uniform Resource Identifier (URI): Generic Syntax*. 2005. `http://tools.ietf.org/html/rfc3986`.

BERNERS-LEE, T. et al. *Tabulator Redux: Writing Into the Semantic Web*. Southampton, UK, 2007.

BIZER, C.; CYGANIAK, R. D2R Server – Publishing Relational Databases on the Semantic Web. In: *5th International Semantic Web Conference*. [S.l.: s.n.], 2006.

BIZER, C.; CYGANIAK, R.; GAUS, T. The rdf book mashup: from web apis to a web of data. In: *The 3rd Workshop on Scripting for the Semantic Web (SFSW 2007), Innsbruck, Austria*. [S.l.: s.n.], 2007.

BIZER, C.; CYGANIAK, R.; HEATH, T. *How to Publish Linked Data on the Web*. [S.l.]: Web-based Systems Group, Freie Universität Berlin, 2007. Http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/.

BIZER, C.; HEATH, T.; BERNERS-LEE, T. Linked Data - The Story So Far. *Int. J. Semantic Web Inf. Syst.*, v. 5, n. 3, p. 1–22, 2009.

BIZER, C.; JENTZSCH, A.; CYGANIAK, R. *State of the LOD Cloud*. 2011. `http://www4.wiwiss.fu-berlin.de/lodcloud/state/`.

BIZER, C.; SEABORNE, A. D2RQ - Treating Non-RDF Databases as Virtual RDF Graphs. In: *ISWC2004 (posters)*. [S.l.: s.n.], 2004.

CLARK, K. G.; FEIGENBAUM, L.; TORRES, E. *Serializing SPARQL Query Results in JSON*. 2008. `http://www.w3.org/TR/rdf-sparql-json-res/`.

CLARK, K. G.; FEIGENBAUM, L.; TORRES, E. *SPARQL Protocol for RDF*. 2008. `http://www.w3.org/TR/rdf-sparql-protocol/`.

D'AQUIN, M. et al. Characterizing knowledge on the semantic web with watson. In: *Evaluation of Ontologies and Ontology-Based Tools: 5th International EON Workshop*. [S.l.: s.n.], 2007.

DAS, S.; SUNDARA, S.; CYGANIAK, R. *R2RML: RDB to RDF Mapping Language*. 2011. `http://www.w3.org/TR/2011/WD-r2rml-20110324/`.

DING, L. et al. Swoogle: a search and metadata engine for the semantic web. In: *Proceedings of the thirteenth ACM international conference on Information and knowledge management*. New York, NY, USA: ACM, 2004. (CIKM '04), p. 652–659. ISBN 1-58113-874-1.

DÜRST, M.; SUIGNARD, M. *RFC 3987 – Internationalized Resource Identifiers (IRIs)*. 2005. `http://www.ietf.org/rfc/rfc3987.txt`.

ERLING, O.; MIKHAILOV, I. *Mapping Relational Data to RDF in Virtuoso*. 2006. `http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOSSQLRDF`.

FIELDING, R. *Architectural Styles and the Design of Network-based Software Architectures*. Tese (Doutorado) — University of California, Irvine, 2000.

FIELDING, R. et al. *RFC 2616 – Hypertext Transfer Protocol – HTTP/1.1*. 1999. `http://tools.ietf.org/html/rfc2616`.

GÖRLITZ, O.; STAAB, S. Federated Data Management and Query Optimization for Linked Open Data. In: VAKALI, A.; JAIN, L. (Ed.). *New Directions in Web Data Management 1*. [S.l.]: Springer Berlin / Heidelberg, 2011, (Studies in Computational Intelligence, v. 331). p. 109–137. ISBN 978-3-642-17550-3.

GRAEFE, G. Encapsulation of parallelism in the volcano query processing system. In: *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 1990. (SIGMOD '90), p. 102–111. ISBN 0-89791-365-5.

HARRIS, S.; SEABORNE, A. *SPARQL 1.1 Query Language*. 2012. `http://www.w3.org/TR/sparql11-query/`.

HARTIG, O.; BIZER, C.; FREYTAG, J.-C. Executing SPARQL Queries over the Web of Linked Data. In: BERNSTEIN, A. et al. (Ed.). *The Semantic Web - ISWC 2009*. [S.l.]: Springer Berlin / Heidelberg, 2009, (Lecture Notes in Computer Science, v. 5823). p. 293–309.

HARTIG, O.; LANGEGGER, A. A Database Perspective on Consuming Linked Data on the Web. *Datenbank-Spektrum*, Springer, v. 14, n. 2, p. 1–10, 2010. ISSN 1618-2162.

HARTIG, O.; ZHAO, J. Publishing and Consuming Provenance Metadata on the Web of Linked Data. *Proc of 3rd Int Provenance and Annotation Workshop*, p. 78–90, 2010.

HEATH, T.; BIZER, C. *Linked Data: Evolving the Web into a Global Data Space*. 1st. ed. [S.l.]: Morgan & Claypool, 2011. 136 p. ISBN 9781608454303.

JARRAR, M.; DIKAIAKOS, M. D. A Query Formulation Language for the Data Web. *IEEE Transactions on Knowledge and Data Engineering*, IEEE Computer Society, 2010.

JENTZSCH, A. et al. Enabling Tailored Therapeutics with Linked Data. In: *Proceedings of the WWW2009 workshop on Linked Data on the Web (LDOW2009)*. [S.l.: s.n.], 2009.

LANGEGGER, A. *A Flexible Architecture for Virtual Information Integration based on Semantic Web Concepts*. Tese (Doutorado) — J. Kepler University Linz, 2010.

LANGEGGER, A.; WÖSS, W. XLWrap – Querying and Integrating Arbitrary Spreadsheets with SPARQL. In: *Proceedings of the 8th International Semantic Web Conference*. Berlin, Heidelberg: Springer-Verlag, 2009. (ISWC '09), p. 359–374. ISBN 978-3-642-04929-3.

LE-PHUOC, D. et al. Rapid prototyping of semantic mash-ups through semantic web pipes. In: *Proceedings of the 18th international conference on World wide web - WWW '09*. [S.l.]: ACM Press, 2009. p. 581–590. ISBN 9781605584874.

LENZERINI, M. Data integration: a theoretical perspective. In: *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. New York, NY, USA: ACM, 2002. (PODS '02), p. 233–246. ISBN 1-58113-507-6.

LORENZO, G. D. et al. Data integration in mashups. *SIGMOD Rec.*, ACM, New York, NY, USA, v. 38, p. 59–66, June 2009. ISSN 0163-5808.

MANOLA, F.; MILLER, E. *RDF Primer*. 2004. `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`. (W3C Recommendation).

MAXIMILIEN, E. M.; RANABAHU, A.; GOMADAM, K. An Online Platform for Web APIs and Service Mashups. *IEEE Internet Computing*, v. 12, n. 5, p. 32–43, set. 2008. ISSN 1089-7801.

MIKHAILOV, I. *Feature: Parameters*. 2009. `http://www.w3.org/2009/sparql/wiki/Feature:Parameters`.

OLIVEIRA, D. E. de; PORTO, F. *QEF User Manual*. [S.l.], September 2010.

OREN, E. et al. Sindice.com: a document-oriented lookup index for open linked data. *Int. J. Metadata Semant. Ontologies*, Inderscience Publishers, Inderscience Publishers, Geneva, SWITZERLAND, v. 3, p. 37–52, November 2008. ISSN 1744-2621.

PINHEIRO, J. C. *Processamento de Consulta em um Framework baseado em Mediador para Integração de Dados no Padrão de Linked Data*. Tese (Doutorado) — Universidade Federal do Ceará, 2011.

PORTO, F. et al. Qef - supporting complex query applications. In: *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2007. (CCGRID '07), p. 846–851. ISBN 0-7695-2833-3.

PRUD'HOMMEAUX, E.; BUIL-ARANDA, C. *SPARQL 1.1 Federated Query*. 2011. `http://www.w3.org/TR/sparql11-federated-query/`.

PRUD'HOMMEAUX, E.; SEABORNE, A. *SPARQL Query Language for RDF*. 2008. `http://www.w3.org/TR/rdf-sparql-query/`.

QUILITZ, B.; LESER, U. Querying Distributed RDF Data Sources with SPARQL. In: *Proceedings of the 5th European semantic web conference on The semantic web: research and applications*. Berlin, Heidelberg: Springer-Verlag, 2008. (ESWC'08), p. 524–538. ISBN 3-540-68233-3, 978-3-540-68233-2.

REDDY, K. B. R.; KUMAR, P. S. Optimizing SPARQL queries over the Web of Linked Data. In: *Proceedings of the International Workshop on Semantic Data Management (SemData 2010), Singapore*. [S.l.: s.n.], 2010.

RIBEIRO, T. de G. et al. Uma Abordagem Baseada em Ontologias para o Projeto de Linked Data Mashups. In: *3ª Conferência Web W3C Brasil*. Rio de Janeiro, RJ, Brazil: [s.n.], 2011.

SACRAMENTO, E. R. et al. Towards Automatic Generation of Application Ontologies. In: *Proceedings of the 25th Brazilian Symposium on Databases – SBBD*. Belo Horizonte, MG, Brazil: [s.n.], 2010. (SBBD '10).

SAUERMANN, L.; CYGANIAK, R. *Cool URIs for the Semantic Web*. [S.l.]: W3C Interest Group Note, 2008. `http://www.w3.org/TR/cooluris/`.

SCHENK, S.; STAAB, S. Networked graphs: a declarative mechanism for sparql rules, sparql views and rdf data integration on the web. In: *Proceedings of the 17th international conference on World Wide Web*. New York, NY, USA: ACM, 2008. (WWW '08), p. 585–594. ISBN 978-1-60558-085-2.

SCHWARTE, A. et al. Fedx: a federation layer for distributed query processing on linked open data. In: *Proceedings of the 8th extended semantic web conference on The semanic web: research and applications - Volume Part II*. Berlin, Heidelberg: Springer-Verlag, 2011. (ESWC'11), p. 481–486. ISBN 978-3-642-21063-1.

SCHWARTE, A. et al. Fedx: optimization techniques for federated query processing on linked data. In: *Proceedings of the 10th international conference on The semantic web - Volume Part I*. Berlin, Heidelberg: Springer-Verlag, 2011. (ISWC'11), p. 601–616. ISBN 978-3-642-25072-9.

SEABORNE, A. *SPARQL 1.1 Query Results CSV and TSV Formats*. 2011. `http://www.w3.org/TR/sparql11-results-csv-tsv/`.

THOR, A.; AUMUELLER, D.; RAHM, E. Data integration support for mashups. In: *Proceedings of the Sixth International AAAI Workshop on Information Integration on the Web*. [S.l.]: Citeseer, 2007. p. 104–109.

TUMMARELLO, G. et al. Sig.ma: Live views on the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, v. 8, n. 4, p. 355 – 364, 2010. ISSN 1570-8268. Semantic Web Challenge 2009; User Interaction in Semantic Web research.

VIDAL, V. M. P. et al. Query Processing in a Mediator Based Framework for Linked Data Integration. *IJBDCN*, v. 7, n. 2, p. 29–47, 2011.

WIEDERHOLD, G. Mediators in the architecture of future information systems. *IEEE Computer*, IEEE, v. 25, n. 3, p. 38–49, March 1992.

APÊNDICE A – PLANOS DE EXECUÇÃO REPRESENTADOS COMO *TEMPLATES* DO QEF

A.1 Plano de Execução relativo à consulta Q1 dos experimentos

```
<?xml version="1.0" encoding="UTF-8"?>
<QEPTemplate xmlns="http://giga03.lncc.br/DIP/WP4/CoDIMS-D"
             xmlns:op="http://giga03.lncc.br/DIP/WP4/CoDIMS-D/Operator"
             xmlns:qep="http://giga03.lncc.br/DIP/WP4/CoDIMS-D/QEP">

  <qep:QEP type="Initial">
    <op:Operator id="1" prod="2" type="">
      <Name>Project</Name>
      <ParameterList>
           <Variables>s, lat, long</Variables>
      </ParameterList>
    </op:Operator>

    <op:Operator id="2" prod="3,4" type="">
      <Name>SetBindJoin</Name>
      <ParameterList>
        <maxActiveThreads>100</maxActiveThreads>
        <blockSize>55</blockSize>
      </ParameterList>
    </op:Operator>

    <op:Operator id="3" prod="0" type="Scan" numberTuples="?">
      <Name>Service</Name>
      <ParameterList>
        <DataSourceName>SparqlEndpoint</DataSourceName>
        <ServiceURI>http://linkedgeodata.arida.ufc.br/sparql</ServiceURI>
        <SPARQLQuery>
          <![CDATA[
          PREFIX owl: <http://www.w3.org/2002/07/owl#>
          select * where {
            ?s owl:sameAs ?geo .
            }
          ]]>
        </SPARQLQuery>
      </ParameterList>
    </op:Operator>

    <op:Operator id="4" prod="0" type="Scan" numberTuples="?">
      <Name>Service</Name>
      <ParameterList>
        <DataSourceName>SparqlEndpoint</DataSourceName>
        <ServiceURI>http://dbpedia.arida.ufc.br/sparql</ServiceURI>
        <SPARQLQuery>
          <![CDATA[
          PREFIX geopos: <http://www.w3.org/2003/01/geo/wgs84_pos#>
          select * where{
            ?geo geopos:lat ?lat ;
                 geopos:long ?long .
          }
          ]]>
        </SPARQLQuery>
      </ParameterList>
    </op:Operator>

  </qep:QEP>
</QEPTemplate>
```

A.2 Plano de Execução relativo à consulta Q2 dos experimentos

```
<?xml version="1.0" encoding="UTF-8"?>
<QEPTemplate xmlns="http://giga03.lncc.br/DIP/WP4/CoDIMS-D"
             xmlns:op="http://giga03.lncc.br/DIP/WP4/CoDIMS-D/Operator"
             xmlns:qep="http://giga03.lncc.br/DIP/WP4/CoDIMS-D/QEP">

  <qep:QEP type="Initial">
    <op:Operator id="1" prod="2" type="">
      <Name>Project</Name>
      <ParameterList>
        <Variables>ds, dg, dgn</Variables>
      </ParameterList>
    </op:Operator>

    <op:Operator id="2" prod="3,4" type="">
      <Name>SetBindJoin</Name>
      <ParameterList>
        <maxActiveThreads>100</maxActiveThreads>
        <blockSize>90</blockSize>
      </ParameterList>
    </op:Operator>

    <op:Operator id="3" prod="0" type="Scan" numberTuples="?">
      <Name>Service</Name>
      <ParameterList>
        <DataSourceName>SparqlEndpoint</DataSourceName>
        <ServiceURI>http://diseasome.arida.ufc.br/sparql</ServiceURI>
        <SPARQLQuery>
          <![CDATA[
          PREFIX diseasome: <http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/>
          select * where{
               ?ds diseasome:possibleDrug ?dg .
          }
          ]]>
        </SPARQLQuery>
      </ParameterList>
    </op:Operator>

    <op:Operator id="4" prod="0" type="Scan" numberTuples="?">
      <Name>Service</Name>
      <ParameterList>
        <DataSourceName>SparqlEndpoint</DataSourceName>
        <ServiceURI>http://dailymed.arida.ufc.br/sparql</ServiceURI>
        <SPARQLQuery>
          <![CDATA[
          PREFIX dailymed: <http://www4.wiwiss.fu-berlin.de/dailymed/resource/dailymed/>
          select * where{
            ?dg dailymed:fullName ?dgn .
          }
          ]]>
        </SPARQLQuery>
      </ParameterList>
    </op:Operator>

  </qep:QEP>
</QEPTemplate>
```

A.3 Plano de Execução relativo à consulta Q3 dos experimentos

```
<?xml version="1.0" encoding="UTF-8"?>
<QEPTemplate xmlns="http://giga03.lncc.br/DIP/WP4/CoDIMS-D"
             xmlns:op="http://giga03.lncc.br/DIP/WP4/CoDIMS-D/Operator"
             xmlns:qep="http://giga03.lncc.br/DIP/WP4/CoDIMS-D/QEP">
```

```
<qep:QEP type="Initial">
  <op:Operator id="1" prod="2" type="">
    <Name>Project</Name>
    <ParameterList>
         <Variables>dgain, dgcf, sen</Variables>
    </ParameterList>
  </op:Operator>

  <op:Operator id="2" prod="3,4" type="">
    <Name>SetBindJoin</Name>
    <ParameterList>
      <maxActiveThreads>100</maxActiveThreads>
      <blockSize>55</blockSize>
    </ParameterList>
  </op:Operator>


  <op:Operator id="3" prod="5,6" type="">
    <Name>SetBindJoin</Name>
    <ParameterList>
      <maxActiveThreads>100</maxActiveThreads>
      <blockSize>55</blockSize>
    </ParameterList>
  </op:Operator>

  <op:Operator id="4" prod="0" type="Scan" numberTuples="?">
    <Name>Service</Name>
    <ParameterList>
      <DataSourceName>SparqlEndpoint</DataSourceName>
      <ServiceURI>http://drugbank.arida.ufc.br/sparql</ServiceURI>
      <SPARQLQuery>
        <![CDATA[
        PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
        select * where{
           ?gdg drugbank:chemicalFormula  ?dgcf .
        }
        ]]>
      </SPARQLQuery>
    </ParameterList>
  </op:Operator>

  <op:Operator id="5" prod="0" type="Scan" numberTuples="?">
    <Name>Service</Name>
    <ParameterList>
      <DataSourceName>SparqlEndpoint</DataSourceName>
      <ServiceURI>http://dailymed.arida.ufc.br/sparql</ServiceURI>
      <SPARQLQuery>
        <![CDATA[
        PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
        PREFIX owl: <http://www.w3.org/2002/07/owl#>
        PREFIX dailymed: <http://www4.wiwiss.fu-berlin.de/dailymed/resource/dailymed/>
        select * where{
          ?dg dailymed:activeIngredient ?dgai .
          ?dgai rdfs:label ?dgain .
          ?dg dailymed:genericDrug ?gdg .
          ?dg owl:sameAs ?sa .
        }
        ]]>
      </SPARQLQuery>
  </ParameterList>
</op:Operator>

<op:Operator id="6" prod="0" type="Scan" numberTuples="?">
  <Name>Service</Name>
  <ParameterList>
```

```
      <DataSourceName>SparqlEndpoint</DataSourceName>
      <ServiceURI>http://sider.arida.ufc.br/sparql</ServiceURI>
      <SPARQLQuery>
        <![CDATA[
        PREFIX sider: <http://www4.wiwiss.fu-berlin.de/sider/resource/sider/>
        select * where{
          ?sa sider:sideEffect ?se .
          ?se sider:sideEffectName ?sen .
        }
        ]]>
      </SPARQLQuery>
    </ParameterList>
  </op:Operator>

  </qep:QEP>
</QEPTemplate>
```

A.4 Plano de Execução relativo à consulta Q4 dos experimentos

```
<?xml version="1.0" encoding="UTF-8"?>
<QEPTemplate xmlns="http://giga03.lncc.br/DIP/WP4/CoDIMS-D"
             xmlns:op="http://giga03.lncc.br/DIP/WP4/CoDIMS-D/Operator"
             xmlns:qep="http://giga03.lncc.br/DIP/WP4/CoDIMS-D/QEP">

  <qep:QEP type="Initial">
    <op:Operator id="1" prod="2" type="">
      <Name>Project</Name>
      <ParameterList>
           <Variables>s, lat, long</Variables>
      </ParameterList>
    </op:Operator>

    <op:Operator id="2" prod="3,4" type="">
      <Name>BindLeftJoin</Name>
    </op:Operator>

    <op:Operator id="3" prod="0" type="Scan" numberTuples="?">
      <Name>Service</Name>
      <ParameterList>
        <DataSourceName>SparqlEndpoint</DataSourceName>
        <ServiceURI>http://linkedgeodata.arida.ufc.br/sparql</ServiceURI>
        <SPARQLQuery>
          <![CDATA[
          PREFIX owl: <http://www.w3.org/2002/07/owl#>
          select * where {
            ?s owl:sameAs ?geo .
            }
          ]]>
        </SPARQLQuery>
      </ParameterList>
    </op:Operator>

    <op:Operator id="4" prod="0" type="Scan" numberTuples="?">
      <Name>Service</Name>
      <ParameterList>
        <DataSourceName>SparqlEndpoint</DataSourceName>
        <ServiceURI>http://dbpedia.arida.ufc.br/sparql</ServiceURI>
        <SPARQLQuery>
          <![CDATA[
          PREFIX geopos: <http://www.w3.org/2003/01/geo/wgs84_pos#>
          select * where{
            ?geo geopos:lat ?lat ;
                 geopos:long ?long .
          }
          ]]>
```

```
        </SPARQLQuery>
      </ParameterList>
    </op:Operator>

  </qep:QEP>
</QEPTemplate>
```

A.5 Plano de Execução relativo à consulta Q5 dos experimentos

```
<?xml version="1.0" encoding="UTF-8"?>
<QEPTemplate xmlns="http://giga03.lncc.br/DIP/WP4/CoDIMS-D"
             xmlns:op="http://giga03.lncc.br/DIP/WP4/CoDIMS-D/Operator"
             xmlns:qep="http://giga03.lncc.br/DIP/WP4/CoDIMS-D/QEP">

  <qep:QEP type="Initial">
    <op:Operator id="1" prod="2" type="">
      <Name>Project</Name>
      <ParameterList>
        <Variables>ds, dg, dgn</Variables>
      </ParameterList>
    </op:Operator>

    <op:Operator id="2" prod="3,4" type="">
      <Name>BindLeftJoin</Name>
    </op:Operator>

    <op:Operator id="3" prod="0" type="Scan" numberTuples="?">
      <Name>Service</Name>
      <ParameterList>
        <DataSourceName>SparqlEndpoint</DataSourceName>
        <ServiceURI>http://diseasome.arida.ufc.br/sparql</ServiceURI>
        <SPARQLQuery>
          <![CDATA[
          PREFIX diseasome: <http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/>
          select * where{
               ?ds diseasome:possibleDrug ?dg .
          }
          ]]>
        </SPARQLQuery>
      </ParameterList>
    </op:Operator>

    <op:Operator id="4" prod="0" type="Scan" numberTuples="?">
      <Name>Service</Name>
      <ParameterList>
        <DataSourceName>SparqlEndpoint</DataSourceName>
        <ServiceURI>http://dailymed.arida.ufc.br/sparql</ServiceURI>
        <SPARQLQuery>
          <![CDATA[
          PREFIX dailymed: <http://www4.wiwiss.fu-berlin.de/dailymed/resource/dailymed/>
          select * where{
            ?dg dailymed:fullName ?dgn .
          }
          ]]>
        </SPARQLQuery>
      </ParameterList>
    </op:Operator>

  </qep:QEP>
</QEPTemplate>
```

A.6 Plano de Execução relativo à consulta Q6 dos experimentos

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<QEPTemplate xmlns="http://giga03.lncc.br/DIP/WP4/CoDIMS-D"
             xmlns:op="http://giga03.lncc.br/DIP/WP4/CoDIMS-D/Operator"
             xmlns:qep="http://giga03.lncc.br/DIP/WP4/CoDIMS-D/QEP">

  <qep:QEP type="Initial">
    <op:Operator id="1" prod="2" type="">
      <Name>Project</Name>
      <ParameterList>
           <Variables>dgain, dgcf, sen</Variables>
      </ParameterList>
    </op:Operator>

    <op:Operator id="2" prod="3,4" type="">
      <Name>BindLeftJoin</Name>
    </op:Operator>


    <op:Operator id="3" prod="5,6" type="">
      <Name>BindLeftJoin</Name>
    </op:Operator>

    <op:Operator id="4" prod="0" type="Scan" numberTuples="?">
      <Name>Service</Name>
      <ParameterList>
        <DataSourceName>SparqlEndpoint</DataSourceName>
        <ServiceURI>http://drugbank.arida.ufc.br/sparql</ServiceURI>
        <SPARQLQuery>
          <![CDATA[
          PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
          select * where{
             ?gdg drugbank:chemicalFormula  ?dgcf .
          }
          ]]>
        </SPARQLQuery>
      </ParameterList>
    </op:Operator>

    <op:Operator id="5" prod="0" type="Scan" numberTuples="?">
      <Name>Service</Name>
      <ParameterList>
        <DataSourceName>SparqlEndpoint</DataSourceName>
        <ServiceURI>http://dailymed.arida.ufc.br/sparql</ServiceURI>
        <SPARQLQuery>
          <![CDATA[
          PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
          PREFIX owl: <http://www.w3.org/2002/07/owl#>
          PREFIX dailymed: <http://www4.wiwiss.fu-berlin.de/dailymed/resource/dailymed/>
          select * where{
            ?dg dailymed:activeIngredient ?dgai .
            ?dgai rdfs:label ?dgain .
            ?dg dailymed:genericDrug ?gdg .
            ?dg owl:sameAs ?sa .
          }
          ]]>
        </SPARQLQuery>
    </ParameterList>
  </op:Operator>

  <op:Operator id="6" prod="0" type="Scan" numberTuples="?">
    <Name>Service</Name>
    <ParameterList>
      <DataSourceName>SparqlEndpoint</DataSourceName>
      <ServiceURI>http://sider.arida.ufc.br/sparql</ServiceURI>
      <SPARQLQuery>
        <![CDATA[
        PREFIX sider: <http://www4.wiwiss.fu-berlin.de/sider/resource/sider/>
```

```
        select * where{
          ?sa sider:sideEffect ?se .
          ?se sider:sideEffectName ?sen .
        }
        ]]>
      </SPARQLQuery>
    </ParameterList>
  </op:Operator>

  </qep:QEP>
</QEPTemplate>
```

A.7 Plano de Execução relativo à consulta Q7 dos experimentos

```
<?xml version="1.0" encoding="UTF-8"?>
<QEPTemplate xmlns="http://giga03.lncc.br/DIP/WP4/CoDIMS-D"
             xmlns:op="http://giga03.lncc.br/DIP/WP4/CoDIMS-D/Operator"
             xmlns:qep="http://giga03.lncc.br/DIP/WP4/CoDIMS-D/QEP">

  <qep:QEP type="Initial">
    <op:Operator id="1" prod="2" type="">
      <Name>Project</Name>
      <ParameterList>
           <Variables>gn, indication</Variables>
      </ParameterList>
    </op:Operator>

    <op:Operator id="2" prod="3,4" type="">
      <Name>Union</Name>
      <ParameterList>
        <useThreads>true</useThreads>
      </ParameterList>
    </op:Operator>

    <op:Operator id="3" prod="0" type="Scan" numberTuples="?">
      <Name>Service</Name>
      <ParameterList>
        <DataSourceName>SparqlEndpoint</DataSourceName>
        <ServiceURI>http://drugbank.arida.ufc.br/sparql</ServiceURI>
        <SPARQLQuery>
          <![CDATA[
          PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
          select * where {
            ?dn drugbank:genericName ?gn ;
                drugbank:indication ?indication.
          }
          ]]>
        </SPARQLQuery>
      </ParameterList>
    </op:Operator>

    <op:Operator id="4" prod="0" type="Scan" numberTuples="?">
      <Name>Service</Name>
      <ParameterList>
        <DataSourceName>SparqlEndpoint</DataSourceName>
        <ServiceURI>http://dailymed.arida.ufc.br/sparql</ServiceURI>
        <SPARQLQuery>
          <![CDATA[
          PREFIX dailymed: <http://www4.wiwiss.fu-berlin.de/dailymed/resource/dailymed/>
          select * where {
            ?dn dailymed:name ?gn ;
            dailymed:indication ?indication .
          }
          ]]>
        </SPARQLQuery>
```

```
      </ParameterList>
    </op:Operator>

  </qep:QEP>
</QEPTemplate>
```

A.8 Plano de Execução relativo à consulta Q8 dos experimentos

```
<?xml version="1.0" encoding="UTF-8"?>
<QEPTemplate xmlns="http://giga03.lncc.br/DIP/WP4/CoDIMS-D"
             xmlns:op="http://giga03.lncc.br/DIP/WP4/CoDIMS-D/Operator"
             xmlns:qep="http://giga03.lncc.br/DIP/WP4/CoDIMS-D/QEP">

  <qep:QEP type="Initial">
    <op:Operator id="1" prod="2" type="">
      <Name>Project</Name>
      <ParameterList>
           <Variables>label, pub_title</Variables>
      </ParameterList>
    </op:Operator>

    <op:Operator id="2" prod="3,4,5,6,7,8,9,10,11,12" type="">
      <Name>Union</Name>
      <ParameterList>
           <useThreads>true</useThreads>
      </ParameterList>
    </op:Operator>

    <op:Operator id="3" prod="0" type="Scan" numberTuples="?">
      <Name>Service</Name>
      <ParameterList>
        <DataSourceName>SparqlEndpoint</DataSourceName>
        <ServiceURI>http://dblp01.arida.ufc.br/sparql</ServiceURI>
        <SPARQLQuery>
          <![CDATA[
          prefix dc:   <http://purl.org/dc/elements/1.1/>
          PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
          select * where {
            ?publication dc:creator ?dblp_researcher ;
                  dc:title ?pub_title .
            ?dblp_researcher rdfs:label ?label ;
            FILTER regex(?label, "^Aab")
          }
          ]]>
        </SPARQLQuery>
      </ParameterList>
    </op:Operator>

    <op:Operator id="4" prod="0" type="Scan" numberTuples="?">
      <Name>Service</Name>
      <ParameterList>
        <DataSourceName>SparqlEndpoint</DataSourceName>
        <ServiceURI>http://dblp02.arida.ufc.br/sparql</ServiceURI>
        <SPARQLQuery>
          <![CDATA[
          prefix dc:   <http://purl.org/dc/elements/1.1/>
          PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
          select * where {
            ?publication dc:creator ?dblp_researcher ;
                  dc:title ?pub_title .
            ?dblp_researcher rdfs:label ?label ;
            FILTER regex(?label, "^Bab")
          }
          ]]>
        </SPARQLQuery>
```

```
      </ParameterList>
    </op:Operator>

    ...

    <op:Operator id="11" prod="0" type="Scan" numberTuples="?">
      <Name>Service</Name>
      <ParameterList>
        <DataSourceName>SparqlEndpoint</DataSourceName>
        <ServiceURI>http://dblp09.arida.ufc.br/sparql</ServiceURI>
        <SPARQLQuery>
          <![CDATA[
          prefix dc:   <http://purl.org/dc/elements/1.1/>
          PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
          select * where {
            ?publication dc:creator ?dblp_researcher ;
                  dc:title ?pub_title .
            ?dblp_researcher rdfs:label ?label ;
            FILTER regex(?label, "^Iab")
          }
          ]]>
        </SPARQLQuery>
      </ParameterList>
    </op:Operator>

    <op:Operator id="12" prod="0" type="Scan" numberTuples="?">
      <Name>Service</Name>
      <ParameterList>
        <DataSourceName>SparqlEndpoint</DataSourceName>
        <ServiceURI>http://dblp10.arida.ufc.br/sparql</ServiceURI>
        <SPARQLQuery>
          <![CDATA[
          prefix dc:   <http://purl.org/dc/elements/1.1/>
          PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
          select * where {
            ?publication dc:creator ?dblp_researcher ;
                  dc:title ?pub_title .
            ?dblp_researcher rdfs:label ?label ;
            FILTER regex(?label, "^Jab")
          }
          ]]>
        </SPARQLQuery>
      </ParameterList>
    </op:Operator>

  </qep:QEP>
</QEPTemplate>
```