UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e de Computação

Talita de Paula Cypriano de Souza

Aplicação de Bancos de Dados Baseados em Grafos no Controle de Redes de Computadores

CAMPINAS
2016

Talita de Paula Cypriano de Souza

Aplicação de Bancos de Dados Baseados em Grafos no Controle de Redes de Computadores

Dissertação apresentada à Faculdade de Engenharia Elétrica e Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestra em Engenharia Elétrica, na Área de Engenharia de Computação.

Orientador: Prof. Dr. Christian Rodolfo Esteve Rothenberg

Coorientador: Prof. Dr. Luciano Bernardes de Paula

Este exemplar corresponde à versão final da dissertação defendida pela aluna Talita de Paula Cypriano de Souza, e orientada pelo Prof. Dr. Christian Rodolfo Esteve Rothenberg

CAMPINAS

2016

Agência(s) de fomento e nº(s) de processo(s): Não se aplica.

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Luciana Pietrosanto Milla - CRB 8/8129

So85a

Souza, Talita de Paula Cypriano, 1990-
Aplicação de banco de dados baseados em grafos no controle de redes de computadores / Talita de Paula Cypriano de Souza. – Campinas, SP : [s.n.], 2016.

Orientador: Christian Rodolfo Esteve Rothenberg.
Coorientador: Luciano Bernardes de Paula.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Bancos de dados. 2. Virtualização de redes. 3. Semântica. 4. Redes de computadores. I. Esteve Rothenberg, Christian Rodolfo,1982-. II. Paula, Luciano Bernardes de. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Applying graph databases in computer networks control plane
Palavras-chave em inglês:
Database
Semantic web
Virtualization
SDN (Software defined networking)
Computer networks
Área de concentração: Engenharia de Computação
Titulação: Mestra em Engenharia Elétrica
Banca examinadora:
Christian Rodolfo Esteve Rothenberg [Orientador]
Leobino Nascimento Sampaio
André Santanchè
Data de defesa: 29-04-2016
Programa de Pós-Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - DISSERTAÇÃO DE MESTRADO

Candidato: Talita de Paula Cypriano de Souza RA: 143063
Data da Defesa: 29/04/2016
Título da Tese: "Aplicação de Bancos de Dados Baseados em Grafos no Controle de Redes de Computadores"

Prof. Dr. Christian Esteve Rothenberg (Presidente, FEEC/UNICAMP)
Prof. Dr. Leobino Nascimento Sampaio (IM/UFBA)
Prof. Dr. André Santanchè (IC/UNICAMP)

Ata de defesa, com as respectivas assinaturas dos membros da Comissão Julgadora, encontra-se no processo de vida acadêmica da aluna.

Aos meus pais e avós

Agradecimentos

Agradeço em primeiro lugar à Deus, por me guiar e me dar forças durante toda a trajetória.

Agradeço aos meus orientadores Dr. Christian Rodolfo Esteve Rothenberg e Dr. Luciano Bernardes de Paula, por todo direcionamento, conselhos, ensinamentos e confiança generosamente dados a mim durante a realização deste trabalho.

Agradeço à minha família por todo amor, apoio e compreensão. Aos meus pais Antonio e Sônia por me ensinarem todos os valores que carrego. Ao meu irmão Janson, pela amizade e incentivo constantes.

Agradeço ao amigo Mateus A. Santos, coautor que contribuiu muito com as publicações que serviram de base para esta dissertação.

Agradeço aos meus amigos do INTRIG e do LCA, por toda a ajuda técnica, amizade e compartilhamento de todos os momentos característicos da vida de pesquisa.

Agradeço aos meus amigos do IFSP-Bragança Paulista, por todas as conversas motivadoras e conselhos.

Agradeço aos professores e colegas da FEEC e do IC, com os quais, durante a realização das disciplinas, pude aprender lições de pesquisa e de vida.

Agradeço a todos os meus amigos por me apoiarem e sempre torcerem por mim.

"Confia ao Senhor as tuas obras, e teus pensamentos serão estabelecidos."
Provérbios 16:3

Resumo

Redes Definidas por Software (SDN) é uma emergente abordagem baseada no desacoplamento do plano de controle do encaminhamento dos dados. Outra tendência atual é a Virtualização das Funções de Rede (NFV), a qual separa as funções dos equipamentos de rede passando a ser executadas em tecnologias de servidor. O plano de controle SDN e o orquestrador NFV, assim como qualquer sistema de controle e gerência de uma rede de computadores, possui a necessidade da representação detalhada e manutenção de modelos de informação sobre sua topologia e os recursos disponíveis. Visando alto desempenho, escalabilidade e facilidade no desenvolvimento de aplicações em rede, em que os dados são altamente conectados e informações topológicas são importantes, os bancos de dados baseados em grafos se apresentam como uma alternativa interessante ao tradicional modelo relacional. A utilização de metadados compatíveis com os padrões da Web Semântica para descrever como os dados são interconectados cada vez mais ganha espaço. Esta dissertação de mestrado explora esse contexto tecnológico e propõe o mapeamento de abstrações de redes de computadores em um banco de dados baseado em grafos, permitindo a obtenção e compartilhamento desses dados entre aplicações e controladores de rede. Para validar a proposta são apresentados três casos de uso: (*i*) mapeamento de um modelo semântico e primitivas para aplicações SDN, (*ii*) suporte de cenários multidomínios e (*iii*) virtualização recursiva no contexto de NFV. Nesta dissertação são apresentados os resultados de avaliações de prova de conceito para cada caso de uso, nos quais um grupo representativo de primitivas foram testadas.

Palavras-chaves:Bancos de Dados Baseados em Grafos, Redes Definidas por Software, SDN, Modelos Semânticos, Virtualização, NFV.

Abstract

Software Defined Networking (SDN) is an emergent approach based on decoupling the control and data planes. Another recent trend is Network Function Virtualization (NFV), which separates the functions from the network equipment and executed on server technologies. Control plane functions of SDN and the NFV orchestrator, like any other control and management system of computer networks, require detailed representation and maintenance of information models about the network topology and the available resources. Towards high performance, scalability, and ease of programmability of network applications, where data is highly connected and rich topological information are important, graph databases appear as an interesting alternative to the traditional relational model. The use of Semantic Web compatible metadata models to describe how data is interconnected grows every day. This dissertation explores these technological trends and proposes the mapping of computer network abstractions to a graph database, allowing the retrieval and sharing of data between network applications and controllers. To validate the proposal, three use cases are presented: (*i*) mapping of SDN primitives following a semantic model, (*ii*) support of multi-domain scenarios, and (*iii*) recursive virtualization in the context of NFV. Evaluation results of the proof of concept implementations for each use case are presented covering a representative set of primitives were tested.

Keywords: Graph Databases, Software Defined Networking, SDN, Semantic Models, Network Function Virtualization, NFV.

Lista de ilustrações

Lista de tabelas

Siglas

ACID Atomicidade, Consistência, Isolamento e Durabilidade. 26

API *Application Programming Interface*. 17, 27, 30, 34, 36, 41, 45, 48, 59, 63, 68

API-IP API *Invocation Point*. 59

APIs *Application Programming Interfaces*. 16, 29, 30, 34

BiS-BiS *Big Switch with Big Software*. 65–68

BSS *Business Support Systems*. 63

C2C *Controller-to-Controller*. 57, 58

CA *Controller Adapter*. 64

CDL *CineGrid Description Language*. 33

CN *Compute Node*. 65

DHT *Distributed Hash Table*. 34

EMS *Element Management Systems*. 63

ETSI *European Telecomunications Standards Institute*. 31

FE *Forwarding Element*. 65

GDB *Graph Database*. 16–19, 26–28, 30, 32, 34, 35, 37–39, 41, 44, 45, 50, 53–56, 59, 62, 63, 65, 68, 69

IL *Infrastructure Layer*. 63, 64

INDL *Infrastructure and Network Description Language*. 32, 33, 69

IP *Internet Protocol*. 46

ISG *Industry Specification Group*. 31

JSON *JavaScript Object Notation*. 41

LLDP *Link Layer Discovery Protocol*. 40

MER Modelo Entidade-Relacionamento. 53

NaaS *Network as a Service*. 35

NDL *Network Description Language*. 21

NF *Network Function*. 64–66

NF-IB *Network Function Information Base*. 64

NFS *Network Functions System*. 63, 64

NFV *Network Function Virtualization*. 16–19, 31, 32, 63, 68, 69

NFV MANO NFV *Management and Orchestration*. 31

NFVI *Network Function Virtualization Infrastructure*. 31

NFVI-PoPs NFV *Infrastructure Points of Presence*. 31

NIB *Network Information Base*. 34, 35

NML *Network Markup Language*. 17, 21, 22, 25, 31–33, 37–41, 43, 44, 46–48, 53, 55, 56, 68, 69

NML-WG *Network Markup Language Working Group*. 22

NOM *Network Object Model*. 34, 35

NOS *Network Operating System*. 29, 30

NoSQL *Not Only SQL*. 16, 25, 26

NOVI *Networking innovations Over Virtualized Infrastructures*. 33

NOVI IM NOVI *Information Model*. 33

OGF *Open Grid Forum*. 22

OL *Orchestration Layer*. 63, 64

ONF *Open Networking Foundation*. 29, 56, 58, 59, 68, 78

ONF-CIM ONF *Common Information Model*. 58, 61, 62

ONOS *Open Network Operating System*. 34

OSI *Open Systems Interconnection*. 46

OSS *Operational Support Systems*. 63

OVF *Open Virtualization Format*. 31

OWL *Web Ontology Language*. 20, 41, 43

RDBM *Relational Database Model*. 25, 36, 38, 53, 54, 68

RDF *Resource Description Language*. 20, 21, 41

REST *Representional State Transfer*. 37

RO *Resource Orchestration*. 63, 64, 66

SDN *Software-Defined Networking*. 16–19, 28–31, 34–41, 45, 46, 53, 56–58, 63, 68, 78

SID *Information Framework*. 31

SL *Service Layer*. 63

SNMP *Simple Network Protocol*. 40

SP *Service Provider*. 63

TOSCA *Topology Orchestration Standard for Cloud Application*. 31

TR *Technical Recommendation*. 56, 58

UML *Unified Modeling Language*. 22, 47, 58

URI *Universal Resource Identifier*. 19, 20, 24, 41

VLAN *Virtual Local Area Network*. 22

VMs *Virtual Machines*. 36

VNF *Virtual Network Function*. 63

VNF-FG VNF-*Forwarding Graph*. 31, 63, 66

VNFs *Virtual Network Functions*. 31

W3C *World Wide Web Consortium*. 19, 20, 41

XML *Extensible Markup Language*. 20, 32

XSD XML *Schema Definition*. 32

Sumário

1 Introdução

Redes definidas por *software*, ou em inglês *Software Defined Networking* (SDN) (KREUTZ *et al.*, 2015), tem se apresentado como uma abordagem inovadora para a construção e operação de redes de computadores com base na identificação e implantação de novas abstrações nos planos de controle e encaminhamento. A abstração mais discutida que foi introduzida pelo modelo SDN é a do conceito de fluxos de pacotes e sua implementação via um protocolo aberto como o OpenFlow (MCKEOWN *et al.*, 2008). Dessa forma, é possível oferecer *Application Programming Interfaces* (APIs) que permitem ao controlador SDN definir (remotamente) o comportamento dos comutadores que suportam o protocolo.

Uma outra importante abstração para o plano de controle de redes SDN que tem recebido menos atenção é a topologia de rede. Na sua versão mais simples, uma topologia pode ser representada como um grafo que interliga os nós da rede usando arestas em função da sua conectividade, seja lógica ou física. Grafos e topologias no geral são (e continuarão sendo) o principal pilar de qualquer abordagem de arquitetura de rede, independentemente de seguir um modelo tradicional com plano de controle totalmente distribuído ou modelos mais centralizados conforme praticado em redes SDN. Seja qual for a abordagem, grafos e suas implementações em estruturas de dados são itens fundamentais para as aplicações de controle que implementam as funções lógicas da rede (ex: geração de árvores mínimas, cálculo de menores caminhos, caminho de recuperação).

Na área de provisionamento de serviços, a tendência que tem ganhado atenção é a Virtualização de Funções de Rede, em inglês *Network Function Virtualization* (NFV) (ROSA *et al.*, 2014), sua proposta é separar as funções dos equipamentos de rede. A arquitetura NFV prevê uma camada de gerenciamento e orquestração, a qual possui também a necessidate manter uma abstração dos elementos da infraestrutura e das funções da rede (MIJUMBI *et al.*, 2015).

No que se trata de armazenamento de tais dados, para torná-los disponíveis aos controladores e aplicações SDN e/ou orquestradores NFV, a categoria emergente de bancos de dados *Not Only SQL* (NoSQL) possui os modelos baseados em grafos, ou em inglês *Graph Databases* (GDB) (ROBINSON *et al.*, 2013). Essa nova categoria de bases de dados prioriza escalabilidade, disponibilidade e menor tempo de resposta. Para o cenário de grande volume de dados e altamente conectados, esses bancos de dados são viáveis. Além disso, os GDBs modelam naturalmente o contexto de topologia de redes (MILLER, 2013), facilitando as tarefas de desenvolvimento de software de aplicações de controle e gerência de redes.

Além do armazenamento, é desejável a adoção de um modelo de dados interoperável com outros domínios, por exemplo, que facilite a troca de dados entre diferentes controladores e automação desses processos. Nesse sentido, padrões de Web Semântica têm sido adotados

nas mais diversas áreas (SHADBOLT *et al.*, 2006). Além da interoperabilidade, tais padrões possuem características de reuso e a de busca aprimorada do dados. Dessa forma, ao invés de se criar um novo modelo de dados com extensas documentações e especificações sobre suas entidades (classes, atributos e relacionamentos), utiliza-se um modelo semântico e o adapta para o contexto do domínio, criando uma extensão.

1.1 Desafios e definição do problema

O objetivo desta pesquisa é aplicar no contexto de gerenciamento de topologias de redes de computadores, o uso de bancos de dados baseados em grafos. Esta dissertação de mestrado foca precisamente no problema de suportar a abstração de uma rede e sua implementação em uma base de dados orientada a grafos. Para isso foram combinados resultados recentes na área de padrões relacionados com Web Semântica para descrição de recursos de redes de computadores *Network Markup Language* (NML) (HAM *et al.*, 2013b) com tecnologias modernas de bases de dados orientadas a grafos (ROBINSON *et al.*, 2013) para sua aplicação no contexto de redes SDN e virtualização NFV.

1.2 Contribuições Científicas

Esse trabalho possui como objetivo utilizar banco de dados baseado em grafos para o armazenamento e consultas de dados e auxiliar na tarefa de gerenciamento de topologias de redes de computadores. Dessa forma, a base de dados se torna a estrutura de dados que mantém o estado da conectividade da rede, e que oferece uma série de primitivas para facilitar o gerenciamento da topologia e a programação das aplicações de controle, por exemplo em contextos SDN e NFV. Além de facilitar o desenvolvimento de aplicações e sua interoperabilidade com diferentes controladores (inclusive em cenários distribuídos e multidomínios), o suporte de uma linguagem semântica para modelar redes SDN abre oportunidades para adicionar raciocínio lógico e técnicas de verificação formal.

Dessa forma, as contribuições desse trabalho são:

1. Aplicação de notação semântica utilizando o NML para redes no contexto de controladores SDN com interfaces a GDBs (Neo4j) e avaliação experimental do desempenho do sistema proposto em protótipo;

2. Mapeamento de primitivas de uma aplicação SDN encontradas na literatura (NetGraph (RAGHAVENDRA *et al.*, 2012)) em consultas utilizando API disponibilizada pelo banco de dados;

3. Identificação de limitações do modelo NML no suporte de primitivas de aplicação controle SDN;

4. Formalização do *parsing* do modelo semântico para o banco de dados e vice-versa.
5. Indexação de dados de topologia e reprodução de primitivas de controladores SDN em cenário de multidomínios;
6. Indexação de dados de infraestrutura, funções de rede e alocação em cenário NFV de virtualização recursiva;
7. Estudo de extensão do modelo semântico para suporte de tabelas de roteamento.

1.3 Organização do Texto

O texto está organizado da seguinte forma:

- **Capítulo 2 - Fundamentação Teórica e Trabalhos Relacionados**: apresenta os fundamentos necessários para a contextualização da pesquisa e os trabalhos relacionados utilizados como motivação para as propostas e casos de uso da pesquisa. Os conceitos das áreas de Web Semântica, armazenamento de dados, SDN e NFV são apresentados como a base da pesquisa, os trabalhos relacionados são discutidos e algumas abordagens são incorporadas na pesquisa;
- **Capítulo 3 - Mapeamento Semântico e Arquitetura**: apresenta uma proposta com uso de modelo semântico para descrever topologia de rede SDN, armazená-la no banco de dados e dar suporte a consultas de um controlador e aplicações. Para tal tarefa, o modelo semântico é analisado e, para lidar com algumas limitações identificadas, o estudo de um extensão é proposto. É proposto uma arquitetura e a formalização do *parsing* do modelo para o banco e vice-versa. Além disso, apresenta a avaliação experimental de um sistema protótipo e os resultados.
- **Capítulo 4 - Cenários de Avaliação**: apresenta dois casos de uso nos cenários de multidomínios SDN e virtualização recursiva NFV. Nas propostas, os dados de topologia são armazenados no GDB e consultados por um controlador e um orquestrador, respectivamente. A primitivas de consultas são escritas em linguagem de consulta do GDB.
- **Capítulo 5**: apresenta as conclusões da pesquisa e discute as direções de trabalhos futuros.

2 Fundamentação Teórica e Trabalhos Relacionados

Neste capítulo são apresentados os fundamentos teóricos e os trabalhos relacionados utilizados para o desenvolvimento da proposta desta pesquisa. A fundamentação começa por modelos semânticos e linguagem de marcação para descrever uma rede, passa por GDB e, contexto SDN e contexto de NFV, os cenários usados como exemplo nesta dissertação. Por fim, os trabalhos desenvolvidos em cada uma das áreas fundamentadas são discutidos e as abordagens incorporadas nesta pesquisa são descritas.

2.1 Web Semântica

Atualmente, com a uma enorme quantidade de dados de diferentes origens, a recuperação de informação é uma tarefa desafiadora (BOUZID; PINATON, 2012). Em um momento, no qual a informação é o verdadeiro valor de uma companhia, a garantia de acesso e compartilhamento dessa informação deve ser majoritária, a qual auxiliará em tomadas de decisões. Nesse cenário, surge a Web Semântica, que tem como objetivo facilitar a automatização do processamento de dados por máquinas através da estruturação dos dados (GOMES; ERVEN, 2011). Segundo o *World Wide Web Consortium* (W3C) (W3C, 2016b), consórcio internacional que desenvolve os padrões da Web, a Web Semântica é a "Web dos dados"(*web of data*), tendo como objetivo permitir que dados sejam compartilhados e reusados entre aplicações, empresas e comunidade. Dessa forma, a ênfase passa a ser nos dados e não nos documentos (SHADBOLT *et al.*, 2006).

Um importante aspecto da Web Semântica é a utilização de *Universal Resource Identifier* (URI)s, identificadores universais para objetos e relacionamentos. A partir da sua utilização, os recursos podem ser interligados (*linked*), referidos ou recuperados por qualquer um, além de permitir que a máquinas processem os dados diretamente (SHADBOLT *et al.*, 2006). O termo “modelo semântico” é adotado neste trabalho, para definir modelos e padrões com abordagem semântica.

Para a definição e descrição de dados em determinados contextos são utilizadas ontologias, que são apresentadas na próxima seção.

2.1.1 Ontologias

Ontologias são artefatos constituídos por um vocabulário específico usados para descrever certa realidade (GOMES; ERVEN, 2011). Ou seja, uma ontologia é como uma rede

conceitual que possui um conjunto de conceitos e noções de um domínio específico (BOUZID; PINATON, 2012). Ontologias são constituídas de classes e hierarquia (subclasses), propriedades, relacionamentos entre classes, restrições e indivíduos. Um padrão recomendado pela W3C para a descrição de ontologias é o *Web Ontology Language* (OWL), uma linguagem de marcação baseada em *Extensible Markup Language* (XML) utilizada para a definição de conceitos e as relações entre eles. A sintaxe completa do OWL está disponível em (HITZLER *et al.*, 2012). Os principais conceitos são:

- ***Class***: conjunto de indivíduos com características comuns.
- ***Individual***: é a instância de uma classe, ou seja, objeto no domínio;
- ***Object Property***: relacionamento entre dois indivíduos;
- ***Data Property***: relacionamento entre indivíduos e valores primitivos (e.g. *string*).

Com a utilização de ontologias a busca não é realizada somente por palavras-chave e sim por conteúdo com significado (GOMES; ERVEN, 2011). Outra vantagem do uso de ontologias é o suporte a regras de inferência, uma forma de raciocínio sobre os dados. Inferência, segundo a W3C, é uma forma de descoberta de novos relacionamentos e verificação de inconsistências (W3C, 2016a).

2.1.2 Metadados

Um dos padrões recomendados pela W3C, para a geração de metadados é o *Resource Description Framework* (RDF). Esse *framework* fornece uma linguagem de representação baseada em triplas na forma: sujeito → predicado → objeto (SHADBOLT *et al.*, 2006). Um objeto de uma tripla pode ser o sujeito de outra, e assim os metadados podem ser relacionados entre si. Para a identificação de cada dado, é usado sua URI. A recomendação com o esquema base do RDF está disponível em (BRICKLEY; GUHA, 2014) e a sua sintaxe RDF/XML está disponível em (GANDON; SCHREIBER, 2014).

A Figura 1 apresenta um exemplo de representação de metadados utilizando RDF que formam um grafo e o Código 2.1 apresenta o mesmo exemplo na sintaxe RDF/XML. Nesse exemplo, o indivíduo identificado pela URI `http://www.w3.org/People/EM/contact#me` possui as *data properties full name* e *personal title*com as *strings* "João Silva"e "Dr."como respectivos valores. Os relacionamentos com outros indivíduos são feitos por meio das *object properties mail box* e *type*. Tanto os indivíduos, quanto *data properties* e *object properties* são representados usando URIs.

Código 2.1 – Exemplo de RDF conforme sintaxe RDF/XML, adaptado de (SHADBOLT *et al.*, 2006)

```
1  <?xml version="1.0"?>
2  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3  xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">
4
5    <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
6     <contact:fullName>João Silva</contact:fullName>
7        <contact:mailbox rdf:resource="mailto:joaosilva@exemplo.com"/>
8        <contact:personalTitle>Dr.</contact:personalTitle>
9    </contact:Person>
10 </rdf:RDF>
```

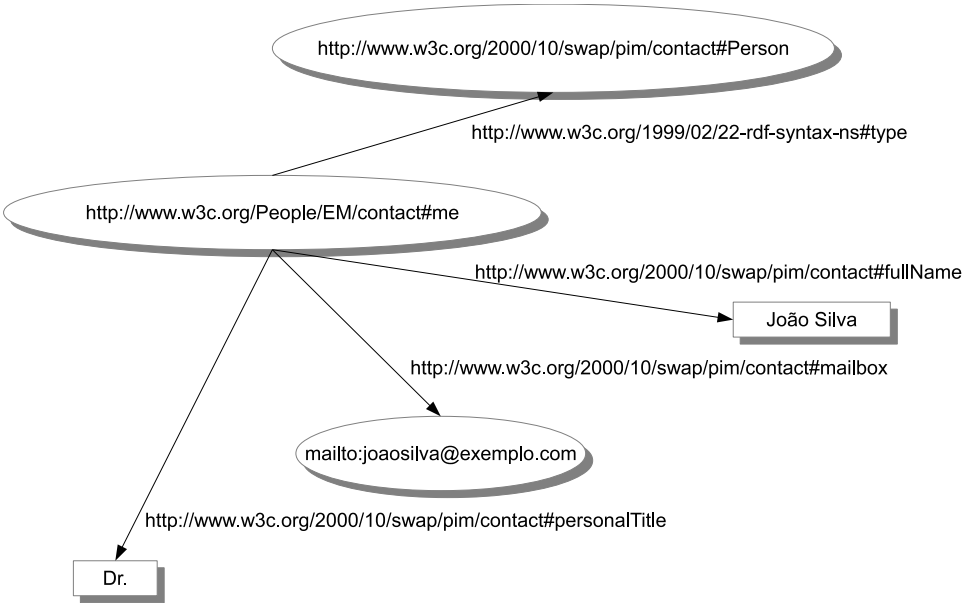

Figura 1 – Exemplo de RDF na representação de grafo (Adaptado de (SHADBOLT *et al.*, 2006))

2.1.3 NML

Um exemplo de linguagem de marcação que utiliza os padrões da Web Semântica para descrever redes de computadores é o NML (HAM *et al.*, 2013b). Trata-se de uma linguagem que descreve os elementos em uma rede do ponto de vista das suas interconexões e elementos interconectados. O NML derivou do *Network Description Language* (NDL), criado nos anos 2000.

No trabalho de van der Ham et al. (HAM *et al.*, 2013b) é feita uma revisão de alguns padrões encontrados na literatura para descrição formal de topologias de redes de computadores. Essa revisão conclui que há pouca pesquisa sobre o assunto e as existentes geralmente são voltadas para grades computacionais (*grids*) e computação em nuvem (*cloud computing*). Nenhum padrão é amplamente adotado, dessa forma a escolha de um formato depende das necessidades do cenário. Devido ao fato de utilizar padrões da Web Semântica e emprego em

recentes projetos relacionados a redes de computadores (e.g. (NOVI, 2016)(ESCALONA *et al.*, 2011)(GROSSO *et al.*, 2011)), o NML foi escolhido para a modelagem neste trabalho.

O NML tem como objetivo descrever redes multicamadas e multidomínios. A especificação dessa linguagem de marcação define que uma rede multicamadas pode ser uma rede virtualizada ou mesmo uma rede utilizando diferentes tecnologias. Com o NML é possível descrever uma topologia de rede, suas capacidades em termos de serviços e sua configuração. O NML tem foco em topologias orientadas à conexão, ou seja, aquelas nas quais o encaminhamento é feito baseado em um fluxo com *labels*, por exemplo *Virtual Local Area Network* (VLAN). Esse modelo também pode ser utilizado para descrever redes físicas ou orientadas a pacotes, entretanto, o seu atual esquema base não contém classes ou propriedades para tratar atributos como degradação de sinal ou tabelas de roteamento (HAM *et al.*, 2013b). Dentro do *Open Grid Forum* (OGF) existe um grupo focado no desenvolvimento e evolução da linguagem, o *Network Markup Language Working Group* (NML-WG).

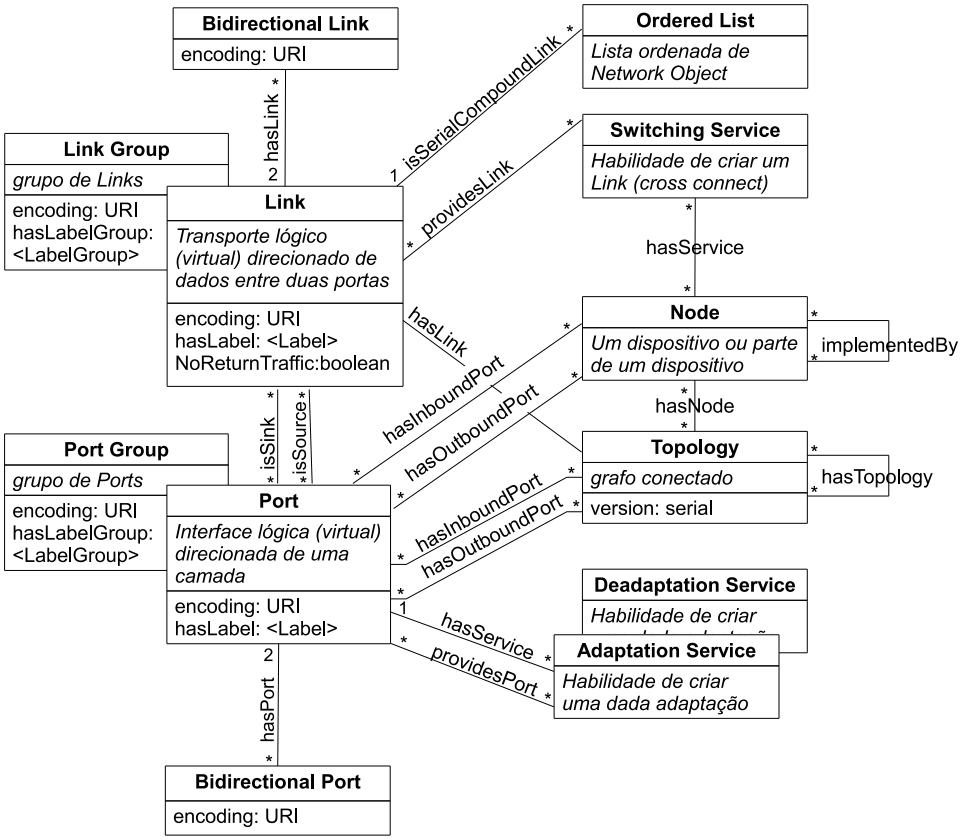

Figura 2 – Diagrama de Classes UML das principais classes do NML *schema*, relacionamentos e suas cardinalidades (Adaptado de (GHIJSEN *et al.*, 2013))

A Figura 2 apresenta o diagrama de classes com as principais classes, relacionamentos e suas cardinalidades do *schema* do NML. Abaixo, essas classes são apresentadas brevemente:

- ***Node***: subclasse de *Network Object*, que define um dispositivo conectado na rede, ou

parte dela, não correspondendo, necessariamente, a uma máquina física.

- ***Port***: subclasse de *Network Object*, que define a conectividade de um *Network Object* com o resto da rede.
- ***Link***: subclasse de *Network Object*, que define o transporte unidirecional de dados a partir de cada uma das suas origens para todos os seus destino, pode se referir a qualquer conexão.
- ***Service***: subclasse abstrata da classe *Network Object*, que descreve uma habilidade da rede, ou seja, como o comportamento pode ser modificado dinamicamente.
- ***Switching Service***: subclasse de *Service*, descreve a habilidade de criar novos *Links* a partir de qualquer *Port* de entrada para qualquer *Port* saída.
- ***Adaptation Service***: subclasse de *Service*, que descreve a habilidade que dados de um para mais *Ports* possam ser embutidos na codificação de outra *Port*, ou seja, descreve uma função de adaptação de multiplexação.
- ***Deadaptation Service***: subclasse de *Service*, que descreve a habilidade dos dados de uma ou mais *Ports* possam ser extraídos a partir da codificação dos dados de outra *Port*, ou seja, descreve um função de adaptação de demultiplexação.
- ***Group***: descreve uma coleção de objetos, no qual, qualquer objeto pode ser parte, inclusive, outro grupo. Um objeto pode ser também parte de múltiplos *Groups*.
- ***Topology***: subclasse de *Group*, que descreve um conjunto de *Network Objects* conectados, ou seja, que é ou é possível criar um transporte de dados entre quaisquer dois *Network Objects* na mesma *Topology*, no caso de não existir restrições políticas, de disponibilidade ou técnicas.
- ***Port Group***: subclasse de *Group*, que representa um conjunto não ordenado de *Ports*.
- ***Link Group***: subclasse de *Group*, que representa um conjunto não ordenado de *Links*.
- ***Bidirecional Port***: subclasse de *Group*, que representa um grupo de duas *Ports* (unidirecionais) ou *Port Groups* que formam uma representação bidirecional de uma porta física ou virtual.
- ***Bidirecional Link***: subclasse de *Group*, que representa um grupo de dois *Links* (unidirecionais) ou *Link Groups* que formam uma representação de um *link* bidirecional.
- ***Location***: referência para uma localização geográfica ou de área.
- ***Lifetime***: intervalo de tempo que os objetos estão ativos que pode ser usado para mudanças na rede, refletir operações dinâmicas, auxiliar com problemas de debug, etc.

- ***Label***: valor específico de tecnologia que distingue um simples stream de dados (um canal) embutido em um *stream* mais largo.
- ***Label Group***: conjunto não ordenado de *Labels*.
- ***Ordered List***: lista ordenada de *Network Objects*. São usados para a relação *isSerialCompoundLink*, uma lista ordenadas de *Links* para descrever um caminho pela rede.

O enfoque do trabalho apresentado nesta dissertação está nas classes: *Node*, *Port* e *Link*, especializações da classe *Network Object* (não exibida na Figura 2) , descritas com mais detalhes abaixo:

- ***Network Object***: seus atributos são: *id* - um persistente e globalmente único URI, *name* - um nome legível e *version* - uma etiqueta de tempo.

  Um *Network Object* pode se relacionar com um ou mais *Lifetimes*, por meio do relacionamento *existsDuring*, com um ou mais *Network Objects*, por meio do relacionamento *isAlias* e por fim, com um *Location* por meio do relacionamento *locatedAt*.

- ***Node***: seus atributos são os mesmos da superclasse: *id* e *name*.

  Um *Node* pode se relacionar com as mesmas classes da superclasse e também com um ou mais *Ports* ou *PortGroups* por meio dos relacionamentos *hasInboundPort* e *hasOutboundPort*, com um mais *Services* do tipo *Switch* por meio do relacionamento *hasService* e com um ou mais *Nodes* por meio do relacionamento *implementedBy*.

- ***Port***: representa uma entidade de transporte lógica em um ponto fixo da rede. Um objeto *Port* é unidirecional, e não corresponde, necessariamente, a uma interface física. Seus atributos são os mesmos da superclasse: *id*, *name* e o adicional *encoding* um identificador para o formato do *streaming* dos dados.

  Uma *Port* pode se relacionar com as mesmas classes da superclasse e também com um ou mais *Links* por meio dos relacionamentos *isSink* e *isSource*, com um ou mais *Services* do tipo *Adaptation* ou *Deadaptation* por meio do relacionamento *hasService* e com um *Label* por meio do relacionamento *hasLabel*.

- ***Link***: uma conexão segmentada e um caminho fim-a-fim são descritas por esse objeto. A composição de conexões em um caminho, e a decomposição de segmentos de uma conexão são descritos por uma relação *isSerialCompoundLink*. Seus atributos são os mesmos da superclasse: *id*, *name* e o adicional *encoding* um identificador para o formato do *streaming* dos dados. Além desses atributos, essa classe tem o adicional *noReturnTraffic* que pode ser `true` ou `false` (padrão).

  Um *Link* pode se relacionar com as mesmas classes da superclasse e também com uma *Ordered List* de *Links* por meio do relacionamento *isSerialCompoundLink*.

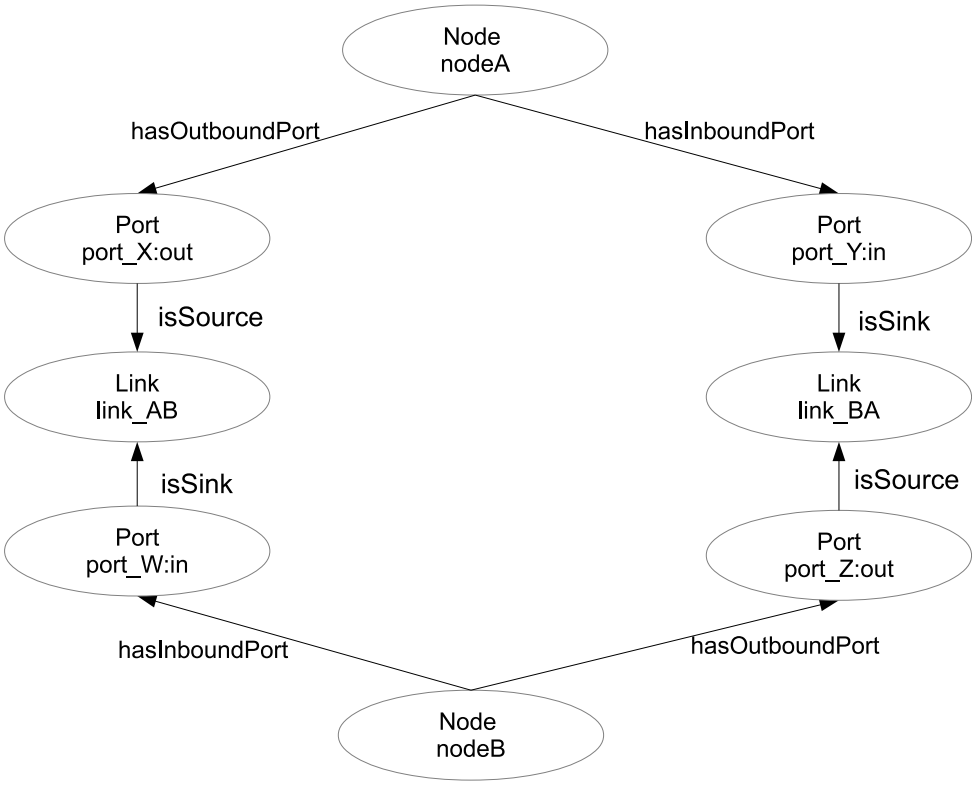

Figura 3 – Exemplo de indivíduos das classes *Node*, *Port*, *Link* e suas relações do modelo NML.

A Figura 3 apresenta um exemplo de indivíduos e suas relações na forma de grafo. O *schema* base do NML permite a definição de extensões do para atender necessidades específicas do domínio, como feito em recentes projetos relacionados a redes de computadores (e.g. (NOVI, 2016)(ESCALONA *et al.*, 2011)(GROSSO *et al.*, 2011)). Essa é uma característica comum de modelos semânticos e é o valor da sua utilização, pois garante o reuso do vocabulário e permite a interoperabilidade entre sistemas. Além disso, a extensão do modelo colabora para sua evolução, serão apresentadas na Seção 2.5 algumas extensões do modelo e no Capítulo 3 o estudo inicial de uma nova extensão a partir das limitações identificadas durante esta pesquisa.

2.2 Armazenamento e Recuperação dos Dados

O tradicional modelo de base de dados relacional, *Relational Database Model* (RDBM), é consolidado, consistente e suas vantagens e desvantagens são bem conhecidas (MILLER, 2013). Entretanto, em algumas tarefas, nas quais a informação topológica e a interconectividade dos dados são importantes, esse modelo apresenta limitações quando comparado com outras abordagens. Nesses casos, a manipulação de dados em um banco relacional pode ser mais complexa e consumir mais tempo. Nesse contexto, uma nova categoria de modelo de banco de dados surgiu, chamada NoSQL. Algumas de suas vantagens são escalabilidade, escalonamento horizontal e ser livre de esquema (NOSQL, 2016).

2.2.1 Banco de Dados Baseado em Grafos

Nos bancos de dados baseados em grafos, GDBs, os quais pertencem à categoria NoSQL, os dados são armazenados como um grafo. A topologia de um grafo G pode ser expressa como $G = (V, E)$, na qual V é o conjunto de vértices e E é o conjunto de arestas. Esse cenário pode ser representado como entidades (vértices) e como essas se relacionam através de suas relações (arestas) (ROBINSON *et al.*, 2013).

Essa abordagem permite a modelagem mais natural de diversos tipos de cenários em diferentes domínios como, por exemplo, a Web Semântica, redes de computadores, mecanismos de recomendação, entre outros. Devido ao crescimento desses domínios, várias soluções têm sido propostas, cada uma delas com suas próprias características e funcionalidades. Mais detalhes podem ser encontrados no trabalho de Jouili e Vansteenberghe (JOUILI; VANSTEENBERGHE, 2013), no qual os autores apresentam a comparação entre importantes implementações desse tipo de banco de dados.

No trabalho de Robinson et al. (ROBINSON *et al.*, 2013) os autores destacam duas características acerca dos modelos de GDB, o “Armazenamento Nativo de Grafo"e o “Processamento Nativo de Grafo". Eles ressaltam que alguns modelos não possuem armazenamento nativo de grafo, ou seja, serializam o grafo em um modelo relacional, orientado a objeto ou outra proposta. O processamento nativo requer que cada elemento possua um apontador para o elemento adjacente e não da indexação de cada elemento.

Entre os modelos de grafo de um GDB estão grafos simples, hipergrafos, grafos aninhados e o grafo de propriedades (*Property Graph*) (PENTEADO *et al.*, 2014). No modelo de grafo de propriedades, adotado na proposta e experimentos desta dissertação, as arestas e os nós contém rótulos e propriedades. A Figura 4 apresenta um exemplo desse tipo de grafo.

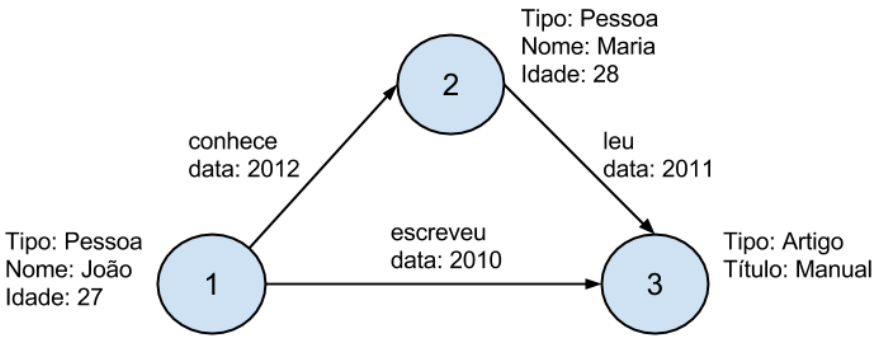

Figura 4 – Exemplo de grafo de propriedades (adaptado de (MILLER, 2013))

O Neo4j [1] (MILLER, 2013) é um GDB desenvolvido pela Neotechnology e possui versão *open-source* e versões comerciais. Algumas de suas características são suporte a transações Atomicidade, Consistência, Isolamento e Durabilidade (ACID), alta disponibilidade e

[1] Neo4j - http://neo4j.com/ (acessado em 15/02/2016)

alta velocidade em consultas. O modelo de grafo que o Neo4j utiliza é o grafo de propriedades. No trabalho Jouili e Vansteenberghe (JOUILI; VANSTEENBERGHE, 2013) é apresentada uma comparação entre diferentes implementações de GDB, na qual o Neo4j se destaca dentre os GDBs testados. Além disso, o Neo4j é caracterizado com armazenamento e processamento de grafo nativo (ROBINSON *et al.*, 2013).

Para a modelagem de dados no Neo4j, os seguintes elementos são considerados:

- **Nó**: é a forma de representar um objeto no banco, pode conter propriedades, relacionamentos e *labels*;
- **Label**: é o tipo do nó, possui um nome e é utilizado para agrupar os nós em conjuntos;
- **Relacionamento**: é a representação de interação entre os nós, possui nome e pode conter propriedades.

Em um RDBM, para recuperar dados com grande interconexão, são necessárias operações complexas do tipo *join*. GDBs foram planejados para resolver esse tipo de problema e apresentarem resultados com alto rendimento (HOLZSCHUHER; PEINL, 2013). O tipo de linguagem de consulta para obter dados de um GDB é chamado *traversal*, pois a consulta "percorre" o grafo através dos nós e suas arestas. Em (HOLZSCHUHER; PEINL, 2013), os autores apresentam uma comparação entre as linguagens de consulta disponíveis para o Neo4j. Em geral, as possibilidades de consulta no Neo4j são: (*i*) Cypher, (*ii*) Gremlin e (*iii*) via API Java com métodos nativos. Cypher é uma linguagem declarativa similiar ao SQL, Gremlin é uma linguagem de consulta fornecida pelo projeto Tinkerpop[2] e a outra possibilidade é executar consultas via API, diretamente da aplicação desenvolvida.

Para a implementação prática desse projeto de pesquisa foi utilizada a linguagem nativa do Neo4j, Cypher, pois essa foi projetada para ser de fácil leitura e entendimento dos desenvolvedores. A linguagem permite escrever consultas que busquem no GDB dados que combinem com determinado padrão (ROBINSON *et al.*, 2013), característica que permitiu aplicar diferentes combinações de acordo com cada primitiva explorada nos casos de uso. Além de consulta, Cypher permite também manipulação dos dados, como por exemplo, atualizações e exclusões (HOLZSCHUHER; PEINL, 2013).

Por exemplo, considere a contagem de conexões de saída em um nó A (do tipo *Node*). A saída de um *Node* é feita pela sua *Port* de saída conectada em um *Link*. Dessa forma, o padrão de busca ficaria da seguinte forma:

```
1. MATCH (n:Node)-[:hasOutboundPort]->(p:Port)-
[:isSource]->(l:Link)
```

[2] Tinkerpop - http://tinkerpop.apache.org/ (acessado em 15/02/2016)

```
2. WHERE n.name="A"
3. RETURN COUNT(l) AS CountOutDegree
```

Na consulta apresentada, o padrão é definido na primeira linha, ou seja, um (*Node*) que tenha um relacionamento [*hasOutboundPort*] com uma (*Port*) que por sua vez tenha um relacionamento do tipo [*isSource*] com um (*Link*). As direções dos relacionamentos são representadas pelo sinal de >. Na segunda linha, a cláusula WHERE especifica o nome do *Node* inicial da busca. A partir deste *Node*, os nós e relacionamentos do banco são percorridos buscando a combinação do padrão. Em seguida é feita a contagem dos *Links* que foram encontrados no caminho, ou seja, equivale a determinar a quantidade de *Links* que a *Port* está conectada.

Além de consultas de somente leitura, a linguagem permite consultas de leitura-escrita no GDB. Por exemplo, considere a criação de relacionamento entre uma *Port* e um *Link*:

```
1. MATCH (a:Port), (b:Link)
2. WHERE a.name="A_out" AND b.name="A_B"
3. CREATE (a)-[r:isSource]->(b)
4. RETURN r
```

No exemplo acima, a primeira linha inicia os tipos dos nós que serão buscados (*Port* e *Link*) e na segunda linha, a cláusula WHERE especifica quais devem ser os nós da busca a partir dos nomes determinados (“A_out” e “A_B”). Após a consulta dos nós, um relacionamento do tipo [*isSource*] é criado entre eles e exibido. Uma documentação completa das funções Cypher está disponível em (NEO4J, 2015).

2.3 Redes Definidas Por *Software*

A aplicação de modelos semânticos e bancos de dados baseados em grafos, explorada pelo trabalho apresentado nesta dissertação, está contextualizado na área de redes de computadores. Na atual infraestrutura de redes, a principal característica é a integração vertical, ou seja, a tomada de decisão do tráfego de rede e o encaminhamento desse tráfego estão acoplados nos componentes (roteadores e *switches)*. Desse modo, a tarefa do operador de rede de programar as políticas desejadas em cada componente, individualmente, e com a tecnologia específica do fabricante, é complexa e de difícil gerenciamento. Além disso, essa característica reduz a flexibilidade e dificulta a inovação e evolução da rede (KREUTZ *et al.*, 2015).

Nesse cenário, surgem as Redes Definidas por *Software* (SDN), um paradigma com o objetivo de superar tais limitações. A sua proposta é separar o controle lógico da rede dos roteadores e *switches* que encaminham os dados, ou seja, os plano de controle do plano de dados. Como consequência da separação, os *switches* se transformam em apenas dispositivos de encaminhamento e o controle é implementado em um controlador centralizado, chamado de

Network Operating System (NOS). Portanto, isso permite a criação de novas abstrações na rede, simplificando o gerenciamento da rede e facilitando a inovação (KREUTZ *et al.*, 2015). Para a "promoção e adoção de SDN, por meio do desenvolvimento de padrões abertos" (ONF, 2016) está a *Open Networking Foundation* (ONF).

A Figura 5 apresenta uma visão geral da arquitetura SDN, a comunicação entre o plano de controle e o plano de dados é feita via APIs, e.g. OpenFlow, assim, o controlador pode instruir o comportamento do *switch* e da mesma forma, ocorre entre as aplicações e o controlador.

Os dispositivos responsáveis pelo encaminhamento de pacotes, possuem instruções para executar ações em cada entrada de pacotes (e.g. encaminhamento para portas específicas, descartá-lo, encaminhamento para o controlador, reescrita de cabeçalho). Tais intruções são definidas por uma interface *southbound* que é também responsável por definir o protocolo de comunicação entre o plano de dados o plano de controle. A ONF definiu como primeiro padrão para essa interface o protocolo OpenFlow (MCKEOWN *et al.*, 2008).

O plano de dados é representado pelos dispositivos de encaminhamento conectados via *wireless* ou cabo físico, ou seja, a representação da infraestrutura da rede. O plano de controle atua como o "cérebro da rede", onde está o controle lógico dos controladores e aplicações. Nesse plano fica o NOS, ou controlador. Por fim, o controlador pode oferecer à aplicações (e.g. algoritmos de roteamento, balanceador de carga, *firewalls*) uma interface para programação, essa é a chamada interface *northbound*.O conjunto dessas aplicações ficam no plano chamado de plano de gerenciamento.

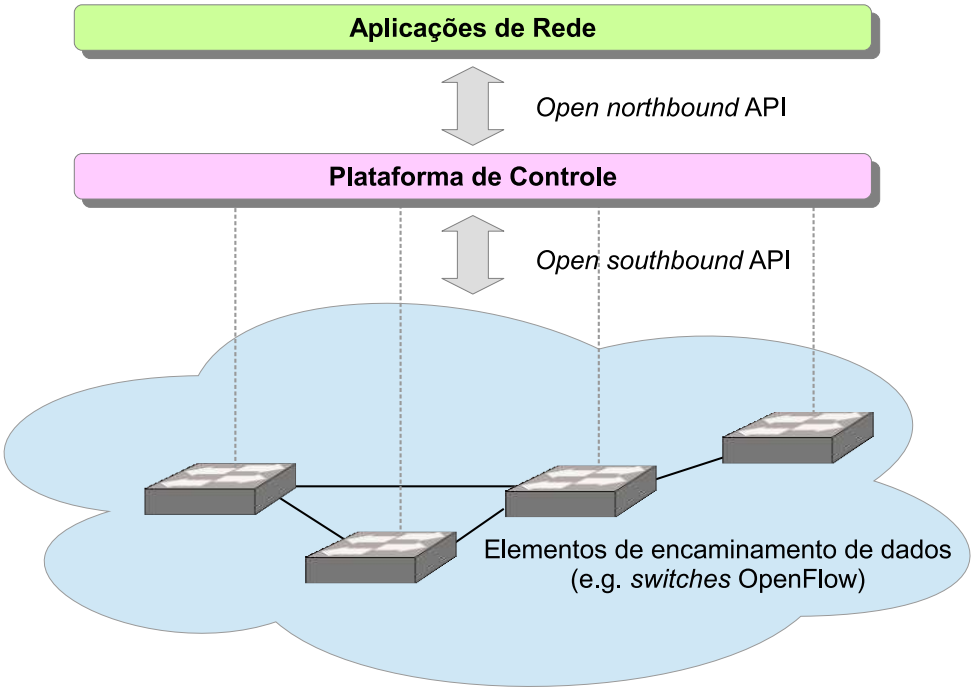

Figura 5 – Visão geral da arquitetura SDN (KREUTZ *et al.*, 2015)

2.3.1 Infraestrutura

A infraestrutura SDN é composta pelos equipamentos comuns de rede, *switches*, roteadores e *middleboxes*. Entretanto, eles são simplesmente dispositivos de encaminhamento, sem embutir qualquer tomada de decisão. Como característica, esses dispositivos devem estar construídos conceitualmente sob uma API padronizadas, e.g. OpenFlow. Os principais elementos para o padrão OpenFlow são o controlador e os comutadores e sua abstração é baseada em fluxos.

2.3.2 Controlador

O objetivo do controlador SDN, NOS, é facilitar o gerenciamento da rede por meio de controle logicamente centralizado. Ele deve dar suporte na geração de configuração de rede baseada em políticas definidas pelo operador da rede. Oferece abstrações, serviços e APIs para desenvolvedores.

Algumas funções consideradas essenciais de um controlador são as de base, tais como, execução de programas, controle de operações I/O, comunicações, proteção e as funções como topologia, estatística, gerenciamento de dispositivos e notificações, encaminhamento de menores caminhos e mecanismos de segurança.

Do ponto de vista de arquitetura, um controlador pode ser centralizado ou distribuído. No centralizado, uma simples entidade gerencia todos os comutadores da rede, em contra partida, o distribuído pode ser fisicamente separado em conjunto de elementos ou ser *cluster* de nós centralizados. Para a arquitetura distribuída, uma vez que informações serão compartilhadas, a compatibilidade e interoperabilidade entre diferentes controladores é feita por meio de interfaces, nesse caso, chamadas de *east/westbound*. Durante a elaboração da arquitetura proposta neste trabalho, essas possibilidades foram consideradas e nos casos de usos foram exploradas.

2.3.3 Aplicações

Uma importante característica do SDN é tornar a rede programável por meio de aplicações de *software* executadas acima do NOS, e assim interagir com as dispositivos do plano de dados (OMNES *et al.*, 2015). Ou seja, as aplicações de rede implementam o controle lógico que será transformado em comandos para serem instalados no plano de dados, determinando o comportamento dos dispositivos de encaminhamento. As aplicações SDN podem relacionadas a engenharia de tráfego, a mobilidade e *wireless*, a monitoramento e segurança entre outras.

Assim, o foco deste trabalho está no contexto do controlador SDN e nas aplicações. Uma vez que o controlador precisa manter uma visão global da rede para atender as aplicações (MIJUMBI *et al.*, 2015), a proposta está no sentido de manter essa visão na forma de um grafo indexado em um GDB com o objetivo de facilitar o acesso aos dados dos diferentes

cenários possíveis em SDN. Além disso, explorar o uso do modelo semântico NML para tais cenários, como será apresentado no Capítulo 3.

2.4 Virtualização das Funções de Rede

Uma tendência que surge na mesma direção de SDN é a Virtualização das Funções de Rede, NFV. Entretanto, NFV está no contexto do provisionamento de serviços de telecomunicações, no qual os operadores de redes precisam desenvolver em dispositivos proprietários e em equipamentos partes das funções. O seu objetivo é separar do equipamento físico as funções de rede que, tradicionalmente, são executadas nele. O benefício está em trazer flexibilidade para os provedores e assim, tornar mais acessíveis suas capacidades de serviços para usuários e outros serviços. Além disso, permite o desenvolvimento/suporte de novos serviços de rede mais rápidos e mais baratos (MIJUMBI *et al.*, 2015). O padrões NFV são definidos pelo *European Telecomunications Standards Institute* (ETSI) no grupo *Industry Specification Group* (ISG) (ETSI, 2015). Comparando o tradicional cenário de provisionamento de serviços com NFV, temos as seguintes características: (*i*) desacoplamento do *software* do *hardware*, (*ii*) desenvolvimento flexível de função de rede e (*iii*) escala dinâmica.

2.4.1 Arquitetura

O *framework* da arquitetura NFV definido pela ETSI ISG pode ser encontrado na documentação em (ETSI, 2014a). Seus componentes são: *Network Function Virtualization Infrastructure* (NFVI), *Virtual Network Functions* (VNFs) e NFV *Management and Orchestration* (NFV MANO), apresentados na Figura 6. Na NFVI estão os recursos de *software* e *hardware* que compõem o ambiente que as VNFs serão implantadas. Tal infraestrutura pode estar separada por localização, cada localização é chamada de NFV *Infrastructure Points of Presence* (NFVI-PoPs) (ETSI, 2014b). Os recursos virtuais são as abstrações de computação, armazenamento e componentes de rede, essas abstrações são feitas pela camada de virtualização. As VNFs são implementações de *software* de uma função de rede, que é executada na NFVI. Por fim, a o NFV MANO é responsável pela orquestração e gerenciamento do ciclo de vida de um recurso de *hardware* ou de *software*. A conexão lógica entre duas VNFs é representada por VNF-*Forwarding Graph* (VNF-FG).

2.4.2 Modelo de Dados

Entre os desafios de NFV está a modelagem de recursos, funções e serviços. Visto que, tais recursos e funções são oferecidos por diferentes entidades e com alta escala de *deploys*, o modelo dos dados deve ser considerado durante todo ciclo de vida do serviço (MIJUMBI *et al.*, 2015). Entre os modelos recomendados pela ETSI estão: *Open Virtualization Format* (OVF), *Topology Orchestration Standard for Cloud Application* (TOSCA), YANG e *Information Fra-*

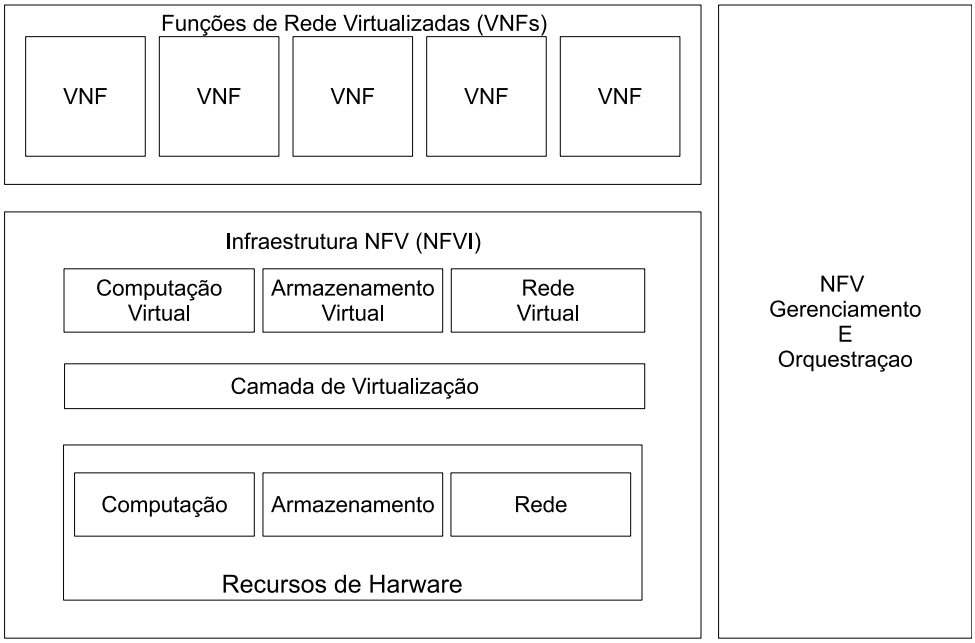

Figura 6 – Visão geral da arquitetura NFV (Adaptado de (ETSI, 2014a)).

mework (SID). A análise realizada no trabalho de Mijumbi et. al (MIJUMBI *et al.*, 2015), tais modelos são baseados em XML ou XML *Schema Definition* (XSD). O objetivo deste trabalho, para o caso de uso de NFV, está voltado para a indexação dos dados de virtualização no GDB. Entretanto, visto que os modelos atuais não exploram as características semânticas, uma possibilidade de trabalho futuro é analisar o modelo NML para tal contexto e propor extensões.

2.5 Trabalhos Relacionados

Nesta seção são apresentados os trabalhos já desenvolvidos nas áreas fundamentadas na seção anterior e as abordagens que foram incorporadas nesta pesquisa.

2.5.1 Modelos Semânticos

Como apresentado na Seção 2.1, o modelo semântico NML é reusável e de fácil extensão, além disso tem sido utilizado em alguns projetos e adaptado conforme o domínio, como será apresentado nessa seção.

Uma das extensões do NML é o modelo *Infrastructure and Network Description Language* (INDL) que permite descrever características de armazenamento e de computação de recursos de infraestrutura, inclusive aplicável em contexto de virtualização (GHIJSEN *et al.*, 2013). O INDL pode ser usado de duas maneiras: (*i*) *stand-alone* ou (*ii*) em combinação com o NML. A ontologia do INDL modela a classe abstrata *Node Component*, uma especialização da classe *Node* do NML, que representa as capacidades de um recurso físico ou virtual. Um *Node Component* pode ser de armazenamento (*Storage Component*), processamento (*Processor*

Component) ou de memória (*Memory Component*).

Um projeto que utiliza essa extensão, INDL, é o *Networking innovations Over Virtualized Infrastructures* (NOVI) (NOVI, 2016), que tem como objetivo federar plataformas para a Internet do futuro. Um dos desafios do projeto é prover interação entre diferentes plataformas. Tanto as requisições quanto os serviços de monitoramento requerem que os diferentes recursos estejam descritos a partir de um modelo. Para isso, o NOVI *Information Model* (NOVI IM), modela a classe *Platform*, subclasse de *Group* (NML) e as especializações *Processing*, *Memory*, *Storage* e *Switching* da classe *Service* (NML).

CineGrid (GROSSO *et al.*, 2011) é uma comunidade multidisciplinar que explora os avanços das infraestruturas de rede e as adapta para o cinema digital. Esse projeto opera como um *testbed* distribuído por vários continentes. Para gerenciar a troca de informação entre os domínios do projeto, a comunidade do CineGrid desenvolveu a ontologia *CineGrid Description Language* (CDL) para descrever sua infraestrutura. CDL foi construído importando classes do NML/INDL, e implementando extensões quando necessário. Sua ontologia é dividida em tipos de serviços e em componentes da infraestrutura, e.g. serviços de armazenamento, processamento de vídeo, serviços de *streaming* e de transcodificação, telas, projetores e etc.

O projeto GEYSERS (ESCALONA *et al.*, 2011) tem como uma de suas inovações a virtualização de infraestruturas ópticas. O modelo de informação GEYSERS é baseado no INDL/NML e provê um modelo para a descrição de dispositivos de redes ópticas, como por exemplo *switches* ópticos. Esse modelo criou a subclasse *Optical Swith Component* de *Node Component* (INDL).

Um trabalho no sentido de propor uma arquitetura descoberta de recursos multidomínio é o de Pittaras et. al (PITTARAS *et al.*, 2012). Os autores apresentam uma arquitetura para combinar os recursos de provedores de múltiplos domínios em uma infraestrutura virtual única. O artigo destaca também os desafios característicos do cenário hetereogêneo da área de descoberta de recursos e serviços distribuídos, tais como, interoperabilidade e consulta de roteamento e infraestrutura dinâmica. Como solução para esses desafios, a arquitetura utiliza o INDL como modelo semântico e como teste de aplicação ela é implementada no projeto NOVI, ambos já apresentados anteriormente. A arquitetura proposta permite ao provedor de recursos a escolha de diferentes níveis de abstração, definidas de acordo com suas políticas de negócio, segurança ou escalabilidade.

Os trabalhos relacionados, apresentados nesta seção, desmonstraram a flexibilidade do modelo semântico NML para os diversos contextos e, além disso, motiva a sua utilização nos trabalhos futuros com modelagem de virtualização e a exploração de múltiplas camadas/abstrações.

2.5.2 Controladores SDN

Em relação ao uso de grafos aplicados em um contexto de SDN, o controlador Onix (KOPONEN *et al.*, 2010) pode ser considerado um trabalho seminal. Onix é uma plataforma de controle desenvolvida por pioneiros na tecnologia OpenFlow e implementa um plano de controle para redes SDN como um problema de sistemas distribuídos se apoiando em dois tipos de bases de dados em função dos requisitos de consistência do estado das aplicações. Onix utiliza grafos para agregar informações de mais baixo nível e distribuir o problema de manutenação das informações entre diferentes controladores. Ele ainda oferece APIs aos desenvolvedores de aplicações de controle, o que inclui uma visão centralizada do estado da rede que simplifica e abstrai detalhes de infraestrutura física. A estrutura de dados utilizada pelo Onix é a *Network Information Base* (NIB), que armazena todas as entidades da topologia da rede usando uma base de dados transacional e informações mais dinâmicas (ex: estatísticas do tráfego nas portas) em uma estrutura de dados distribuída do tipo *Distributed Hash Table* (DHT), mantida em memória apenas como garantia de consistência fraca/eventual. Além disso, a API da NIB oferece funções de consulta, criação, exclusão e acesso a atributos de entidades, notificações de mudanças, sincronização, configuração e importação. As classes padrão da NIB são: super classe *Node* e as especializações *Network*, *Host* e *Forwarding Engine*, *Forwarding Table*, *Port*, *Link*.

Um recente controlador SDN *open-source* que segue os princípios de projeto do Onix é o *Open Network Operating System* (ONOS) (BERDE *et al.*, 2014). ONOS desenvolveu um primeiro protótipo usando um GDB distribuído, Titan, para armazenar o estado da rede e foi movido para um segundo protótipo usando um modelo simplificado com estruturas de dados otimizadas e dados processados em memória por motivo de performance.

O uso de grafos em SDN também é considerado no trabalho de Pantuza et al. (PANTUZA *et al.*, 2014) para permitir que módulos de um controlador possam obter informações sobre a topologia da rede. O artigo ainda apresenta experimentos que mostram o suporte à representação dinâmica da rede. Em especial, os autores implementaram uma árvore geradora de custo mínimo que é mantida em tempo real sobre o grafo da rede. Para implementação do trabalho, os autores utilizam o POX, um controlador baseado em Python. O principal módulo utilizado é *Topology*, responsável por manter um dicionário de objetos representando as entidades da rede, chamado de *Network Object Model* (NOM). Na proposta de arquitetura, os autores integram o módulo com eventos para descoberta de dispositivos na rede, criação, atualização, exclusão, execução de algoritmos e recuperação dos dados armazenados no grafo.

O trabalho supracitado é similar ao NetGraph (RAGHAVENDRA *et al.*, 2012), pois além de suportar atualizações periódicas do estado da rede, a biblioteca NetGraph também oferece resultados de consultas que podem ser utilizados por um controlador SDN. Uma característica peculiar do NetGraph é o pré-cálculo de determinadas operações para otimizar o tempo de consultas. Por exemplo, menores caminhos entre pares de nós da rede são pré-calculados de forma parcial, o que pode ser utilizado por algoritmos de roteamento. Dessa forma, a biblioteca

NetGraph, implementa duas principais funcionalidades: (1) consultar a topologia de rede incluindo nós e estado de links para manter um grafo de rede atualizado e (2) computar consultas do grafo e retornar os resultados da consulta de uma forma que possa ser utilizada por outros módulos para prover virtualização de redes, *Network as a Service* (NaaS).

Outra interessante proposta que utiliza a abstração de grafos em SDN é a do trabalho de Lauer et. al (LAUER *et al.*, 2013). Nesse trabalho, os autores propõem um método para fazer troca de informação entre controladores (*Control Planes*). Para tal troca, o controlador exporta um subgrafo, ou seja, uma abstração da sua rede com conjunto de recursos. Os autores chamam o subgrafo de "*showed subgraph*". O controlador representa toda a topologia da rede, tais como, estado de *links*, políticas, métricas e metadados em vértices, relacionamentos e atributos de um grafo. Na implementação do trabalho, eles fazem uso do GDB DEX.

Os trabalhos apresentados nessa seção possuem propostas no sentido do uso de abstração e funcionalidades de grafos em controladores SDN. Uma sumarização dessas propostas é apresentada na Tabela 1. Em todos esses trabalhos os modelos de dados não utilizam padrões condizentes com as características da Web Semântica. Cada trabalho utiliza seu próprio modelo e documentação. Dessa forma, o trabalho desta dissertação explora as características de reuso e interoperabilidade como é apresentado no Capítulo 3. Outra característica explorada nos casos de uso que serão apresentados no Capítulo 4 é a utilização de GDB para modelagem e consulta dos dados, explorando suas funcionalidades nativas.

Trabalho	**Modelo de Dados**	**Banco de Dados**
Onix (KOPONEN *et al.*, 2010)	NIB	Base de dados transacional
POX Adaptado (PANTUZA *et al.*, 2014)	NOM	Não aplica
NetGraph (RAGHAVENDRA *et al.*, 2012)	XML	Não aplica
ONOS (BERDE *et al.*, 2014)	Modelo próprio	GDB e Estrutura de Dados Otimizada
Showed Subgraphs (LAUER *et al.*, 2013)	Modelo próprio	GDB DEX

Tabela 1 – Propostas de Controladores SDN

2.5.3 Armazenamento de Dados

Dada a natureza dos dados de topologia de rede, os bancos de dados considerados para este trabalho são os orientados a grafos, GDB.

O trabalho de Jouili e Vansteenberghe (JOUILI; VANSTEENBERGHE, 2013), já citado na Seção 2.2, realiza um *benchmarking* dos GDBs Neo4j, Titan, OrientDB e DEX. As tarefas realizadas pelo *benchmarking* foram de: (*i*) inserção de dados, (*ii*) consultas (*traversals*) de cálculo de *shortest path* e exploração de vizinhança e (*iii*) requisições intensivas (consultas e atualização) paralelas. A arquitetura foi implementada de maneira distribuída (*master-slave*), com o objetivo de simular requisições concorrentes ao banco de dados e analisar sua performance. Como resultado final, o Neo4j apresentou um melhor desempenho para os *traversals*, apesar disso, sua performance não para as consultas de leitura e escrita caiu consideravelmente

e as bases Titan e DEX ficaram a frente. Ainda sim, o Neo4j foi a escolha de implementação desse trabalho, devido a importância das consultas de *traversal* no contexto SDN, tarefa que o Neo4j se destacou. Além disso, a comunidade e suporte são bem ativos e existe quantidade importante de documentação disponível.

Na direção de *benchmarking*, como já citado na Seção 2.2, os autores Holzschuher e Peinl (HOLZSCHUHER; PEINL, 2013) apresentam os resultados de testes com as possíveis linguagens de consulta do Neo4j, Cypher e Gremlin, com consultas no RDBM MySQL. Os autores focaram na ánalise de desempenho, compreensibilidade e linhas de código em um cenário de rede social. Na comparação de legibilidade e eficiência do código, apesar do código Java usando API do Neo4j possuir menos linhas de código, o Cypher apresentou maior facilidade de leitura e de manutenção do código. Na comparação das linguagens Cypher e Gremlin, o Gremlin apresentou melhor desempenho em alguns tipos de consultas, entretanto quando as consultas são mais complexas, ou seja, com maior número de nós e relacionamentos, se torna mais difícil a escrita e leitura do código do que no Cypher. Os autores observaram também que o Neo4j apresenta melhor desempenho das consultas quando o número de arestas é maior, comparado com o MySQL.

Um outro trabalho relacionado importante é o dos autores Soundararajan e Kakaraddi (SOUNDARARAJAN; KAKARADDI, 2014) que utilizam o banco de dados baseado em grafo Neo4j para tarefas de auditoria em *cloud*. Eles implementam consultas na linguagem *Cypher* para solucionar essas tarefas, linguagem que se mostrou intuitiva e extensível. Entre as consultas criadas estão: análise de risco, para determinar as *Virtual Machines* (VMs) que seriam afetadas de uma rede e *datastore* em caso de uma queda de energia; reporte simples, para verificar qual é o arranjo de armazenamento que estão sendo usados pelas VMs e comparação de inventário, para verificar se duas hierarquias são equivalentes, entre outras. Esse trabalho, apresentou a viabilidade do uso do Neo4j para o contexto de *cloud*, que pode ser estendido para virtualização de serviços de rede, como será apresentado no caso de uso do Capítulo 4.

3 Mapeamento Semântico e Arquitetura

Conforme os conceitos apresentados e os trabalhos já desenvolvidos apresentados no Capítulo 2, a proposta desta pesquisa é a indexação dos dados da topologia de rede SDN no banco conforme o modelo semântico NML. O objetivo é auxiliar um controlador em suas primitivas (RAGHAVENDRA *et al.*, 2012), tanto a representação da topologia da rede quanto a modelagem dos dados seguindo o modelo semântico utilizam grafos. A proposta desse caso de uso vislumbra facilitar o processamento da primitivas, por meio de funções nativas oferecidas pelo banco de dados, e garantir as vantagens do uso de um modelo semântico, tais como reuso, interoperabilidade e inferência. A seguir são detalhadas as características da arquitetura proposta, bem como o estudo inicial de uma extensão do modelo semântico e uma avaliação experimental.

3.1 Arquitetura

Para atender aos objetivos deste trabalho, a arquitetura proposta prevê o suporte ao modelo semântico escolhido, por meio da implementação de um método de *parsing* responsável pela "tradução" dos dados da topologia. Além disso, o banco de dados deve ser facilmente integrado ao controlador SDN que oferece as primitivas para aplicações de controle via interfaces *northbound* (e.g. *Representional State Transfer* (REST)) e manter o estado da rede SDN a partir da comunicação com *switches* pelas interfaces *southbound* (e.g. OpenFlow, NETCONF). Vislumbra-se ainda a comunicação entre diferentes controladores a partir de uma interface *east-west*, permitindo receber ou transmitir descrições da rede com o uso de um modelo semântico. Esse método abre novas oportunidades de operação de redes SDN, já que um controlador poderia mapear o relacionamento entre múltiplas fontes de informação e selecionar ações apropriadas para oferecer novos serviços (PULKKINEN *et al.*, 2012). No Capítulo 4 será descrito o caso de uso considerando o cenário de multidomínios, no qual um controlador exporta diferentes visões da topologia para transmitir a outro controlador.

A Figura 7 apresenta a arquitetura proposta, onde aplicações externas ou internas a um controlador SDN podem realizar consultas em um GDB. Isso permite obter informações referentes às características e à situação da rede de forma centralizada, ao invés de cada aplicação realizar sua consulta separadamente. Esse banco de dados, por sua vez, recebe informações atualizadas por meio de módulos que utilizam interfaces com o plano de dados ou com outros controladores (e.g. gerenciador de topologias).

O GDB oferece ao controlador SDN informações resultantes de primitivas do tipo: o menor caminho entre dois nós, a contagem do grau de conectividade de um determinado nó, seus vizinhos, entre outras, além da possibilidade de criação de outras primitivas combinando

as já existentes (e.g. todos os caminhos via diferentes camadas entre máquinas virtuais em *hosts* com interfaces 10 G). Essas informações ajudam o controlador SDN e suas aplicações na tomada de decisões em relação à atuação na rede de forma precisa e em tempo viável , como apresentado nas próximas seções.

Como mencionado anteriormente no Capítulo 2, este trabalho adota o Neo4j como banco de dados de implementação, justificada pelo seu desempenho em comparação com outras implementações de GDB (JOUILI; VANSTEENBERGHE, 2013). Em relação ao modelo semântico, o uso de NML permite trabalhar com representações específicas para redes de computadores sem preocupação com detalhes de infraestrutura. Assim como os padrões da Web Semântica, o NML é flexível permitindo a criação de novas extensões.

3.2 Modelagem dos Dados

A modelagem é uma tarefa de abstração e seu objetivo é transformar um problema de domínio específico para um cenário que em posssa ser estruturado e manipulado. Tanto para um RDBM quanto para um GDB a tarefa é necessária, entretanto a representação em grafo traz algumas vantagens, entre elas, a modelagem é mais natural e o modelo lógico é muito mais próximo do modelo físico, quando comparado à abordagem tradicional (ROBINSON *et al.*, 2013).

Para atender à proposta deste trabalho, na utilização do modelo semântico NML e na indexação dos dados no GDB, o modelo lógico é apresentado na Figura 8. Cada nó do tipo *Node* se relaciona com nós do tipo *Port*, que podem ser do tipo de entrada ou saída, o que é determinado pelo tipo e direção do relacionamento, respectivamente *hasInboundPort* e *hasOutboundPort*. A representação de conexão entre dois nós do tipo *Node* é feita por relacionamentos

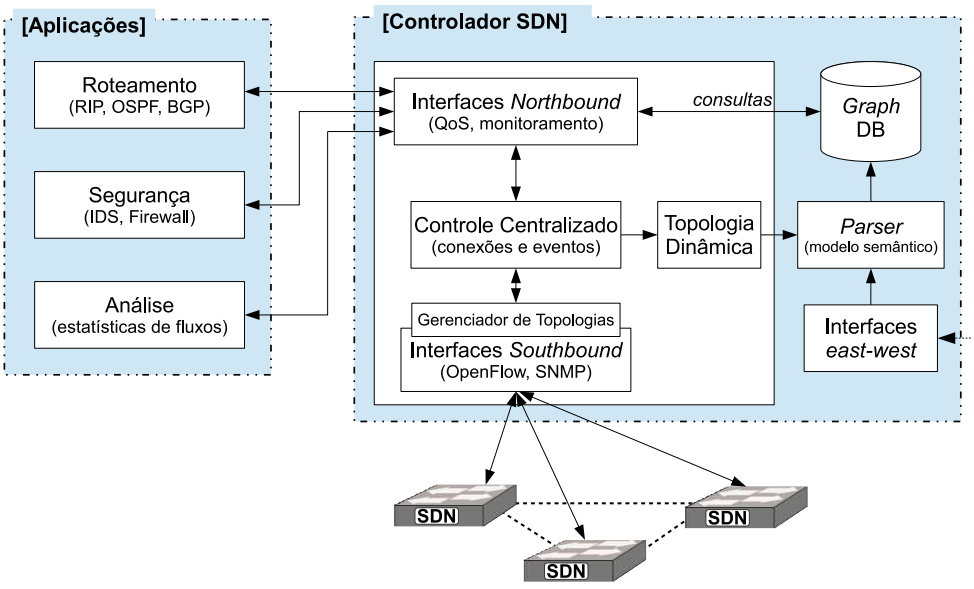

Figura 7 – Arquitetura proposta integrando GDB com suporte ao Modelo Semântico

entre suas *Ports* com um nó do tipo *Link*, devido a este ser direcional (HAM *et al.*, 2013a), e os relacionamentos são no sentido do fluxo, ou seja, *isSource* (origem) e *isSink* (destino).

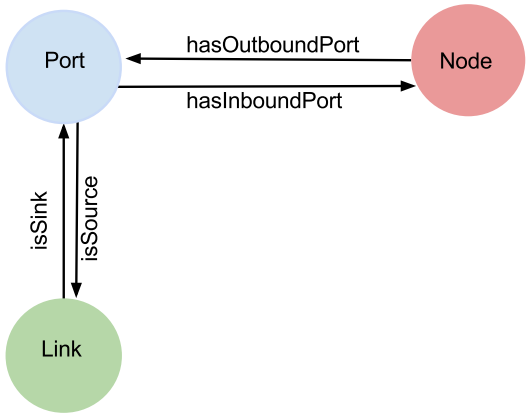

Figura 8 – Modelo lógico dos dados na representação de grafo, baseado no NML

3.3 Análise das Primitivas

Para atender o objetivo desta pesquisa, de auxiliar a tomada de decisão de um controlador SDN, primitivas da biblioteca NetGraph (RAGHAVENDRA *et al.*, 2012) foram analisadas em relação a compatibilidade com o modelo semântico e com o banco de dados, respectivamente, NML e Neo4j. As primitivas e os resultados da análise de compatibilidade são apresentados na Tabela 2. Foi considerado que uma primitiva é suportada pelo modelo semântico quando esta pôde ser respondida a partir dos atributos e relacionamentos de um grafo modelado no NML, considerando o *schema* apresentado na Figura 2, no Capítulo 2. Da mesma forma, foi considerado que uma primitiva tem suporte do GDB quando esta pôde ser respondida por meio de uma ou mais consultas na linguagem *Cypher*. A última coluna da tabela se refere ao tipo da primitiva, escrita ou leitura.

Primitiva	**Descrição**	**Modelo Semântico**	**GDB**	**Leitura/ Escrita**
`setEdgeWeight`	Atribuição de peso a uma aresta	Não	Sim	E
`getEdgeWeight`	Obtenção de peso de uma aresta	Não	Sim	L
`countInDegree`	Grau de entrada de um nó	Sim	Sim	L
`countOutDegree`	Grau de saída de um nó	Sim	Sim	L
`countNeighbors`	Contagem de vizinhos de um nós	Sim	Sim	L
`ComputeMST`	Cálculo de *Minimum Spanning Tree*	Sim	Sim	L
`computeAPSP`	Cálculo de todos os pares de menores caminhos	Sim	Sim	L
`computeSSSP`	Cálculo de menores caminhos a partir de um nó	Sim	Sim	L
`doesRouteExist`	Verificação de rota entre dois nós	Sim	Sim	L
`computeKSSSP`	Cálculo de k menores caminhos entre dois nós	Sim	Sim	L
`delete`	Exclusão de nó	Sim	Sim	E
`insert`	Inserção de nó	Sim	Sim	E

Tabela 2 – Compatibilidade das Primitivas da literatura com o Modelo Semântico e o GDB

As primitivas `setEdgeWeight` e `getEdgeWeight`, respectivamente, atribuem custo para determinada conexão entre dois *Nodes* e obtêm esse custo. Essas primitivas não são suportadas pelo modelo semântico, pois o *schema* do NML não possui o atributo de custo de conexão. Essa limitação foi resolvida com o estudo de uma extensão do modelo, como será apresentado na Seção 3.5, na qual é acrescentado um atributo de custo (*cost*) à uma entidade *Link*. No banco de dados, o modelo de grafo de propriedades, se apresentou compatível com a extensão proposta.

As primitivas `countInDegree`, `countOutDegree` e `countNeighbors` que calculam a quantidade de conexões de entrada e de saída de um *Node* e seus vizinhos, são suportadas tanto pelo modelo semântico quanto pelo banco de dados. Nos experimentos, para a contagem do grau de um *Node*, foi calculada a quantidade de *Links* ligados às portas (*Port*) de entrada e de saída de um nó e para a contagem de vizinhos, é a mesma implementação, considerando os *Nodes* conectados.

As primitivas `computeMST`, que gera a *Minimum Spanning Tree* a partir de uma origem, e `computeSSSP` (*Single Source Shortest Path*), que retorna o menor caminho de um *Node* para todos os outros, utilizaram a mesma função nativa da linguagem *Cypher* chamada *shortestPath*. Esse recurso foi usado também para encontrar o menor caminho de cada par de *Nodes* da primitiva `computeAPSP` (*All Pair Shortest Path*). Outro recurso nativo do *Cypher* utilizado foi o `allShortestPath` para a primitiva `computeKSSSP` (*k Single Source Shortest Path*), que encontra *k* menores caminhos entre dois *Nodes*. Para essas primitivas de menores caminhos foi constatado que o modelo semântico possui suporte, visto que sua modelagem gera um grafo que naturalmente permite essas operações. Nesse sentido, a primitiva `doesRouteExist` que verifica a existência de rota entre dois *Nodes* é suportada pelo modelo semântico e pelo banco de dados.

As primitivas de escrita `insert` e `delete` são suportadas pelo banco de dados e pelo modelo semântico. A inserção de um *Node* no banco de dados realiza a inserção de duas *Ports* e os relacionamentos entre eles, conforme modelagem apresentada na seção anterior. Da mesma forma, uma exclusão remove *Ports* e *Links* que o *Node* está relacionado, bem como seus relacionamentos.

Para a implementação, primitivas relacionadas com o grau de um nó podem ser obtidas com o uso do protocolo *Link Layer Discovery Protocol* (LLDP), o que já é realizado por implementações de controladores SDN como o *OpenDaylight* [1]. O custo de uma conexão entre dois nós pode ser identificado a partir da latência ou da largura de banda utilizada. O *OpenFlow* permite obter esse tipo de métrica através de mensagens "*Echo request/reply*", porém há melhor precisão ao se utilizar métodos específicos, como o protocolo *Simple Network Protocol* (SNMP).

[1] OpenDaylight - http://www.opendaylight.org/ (acessado em 25/01/2016)

3.4 *Workflow*

Para a implementação da proposta de indexação de dados conforme modelagem semântica é necessário realizar *parsing* dos dados, para as seguintes conversões:

- Modelo Semântico → Banco de Dados: a partir do modelo semântico, inserir corretamente os dados no banco de dados.
- Banco de Dados → Modelo Semântico: o caminho inverso, a partir do grafo indexado no banco de dados, gerar os metadados do modelo semântico em formato definido.

O *workflow* do *parsing* para tais tarefas está detalhado na Figura 9, que apresenta a trajetória desde a entrada dos dados da topologia, a sua indexação no GDB e as consultas para aplicações SDN. As etapas apresentadas são:

1. **Importação dos dados e modelagem semântica:** as informações da topologia obtidas por um gerenciador de topologia, e.g. *Topology Manager* OpenDaylight, são pré-processadas e transformadas em um modelo semântico conforme um *framework* escolhido, e.g. RDF. Essa etapa será detalhada na Subseção 3.4.1.
2. **Geração e inserção do grafo no GDB:** a partir do modelo semântico, o grafo é gerado e inserido no GDB, por meio de uma API previamente programada ou *queries* de inserção. Essas tarefas serão detalhadas na Subseção 3.4.2.
3. **Consultas e atualizações:** as aplicações consultam o GDB, por meio das primitivas apresentadas na Seção 3.3. A Subseção 3.4.3 detalhará essas tarefas.

Apesar do *workflow* não estar vinculado a tecnologias específicas, todos os detalhes a seguir estão relacionados ao modelo NML e GDB Neo4j.

3.4.1 Importação dos Dados e Modelagem Semântica

Para a gerar o modelo semântico, o padrão escolhido foi o OWL RDF/XML, conforme a sintaxe apresentada na documentação do NML (HAM *et al.*, 2013a). Importante ressaltar, que o modelo semântico poderia ser descrito em outros padrões determinados pela W3C, e.g. *JavaScript Object Notation* (JSON). A documentação do *schema* do NML define também o *namespace* utilizado, ou seja, todas as classes, relacionamentos e atributos estão definidos em `http://schemas.ogf.org/ nml/2013/05/base#`. Os identificadores (URIs) das instâncias utilizados nos exemplos apresentados a seguir, estão em conformidade com a *Global Networks Identifiers*, definido em (DIJKSTRA; HAM, 2013). As instâncias estão definidas como `urn:ogf:network:experiment.mt:2015:`.

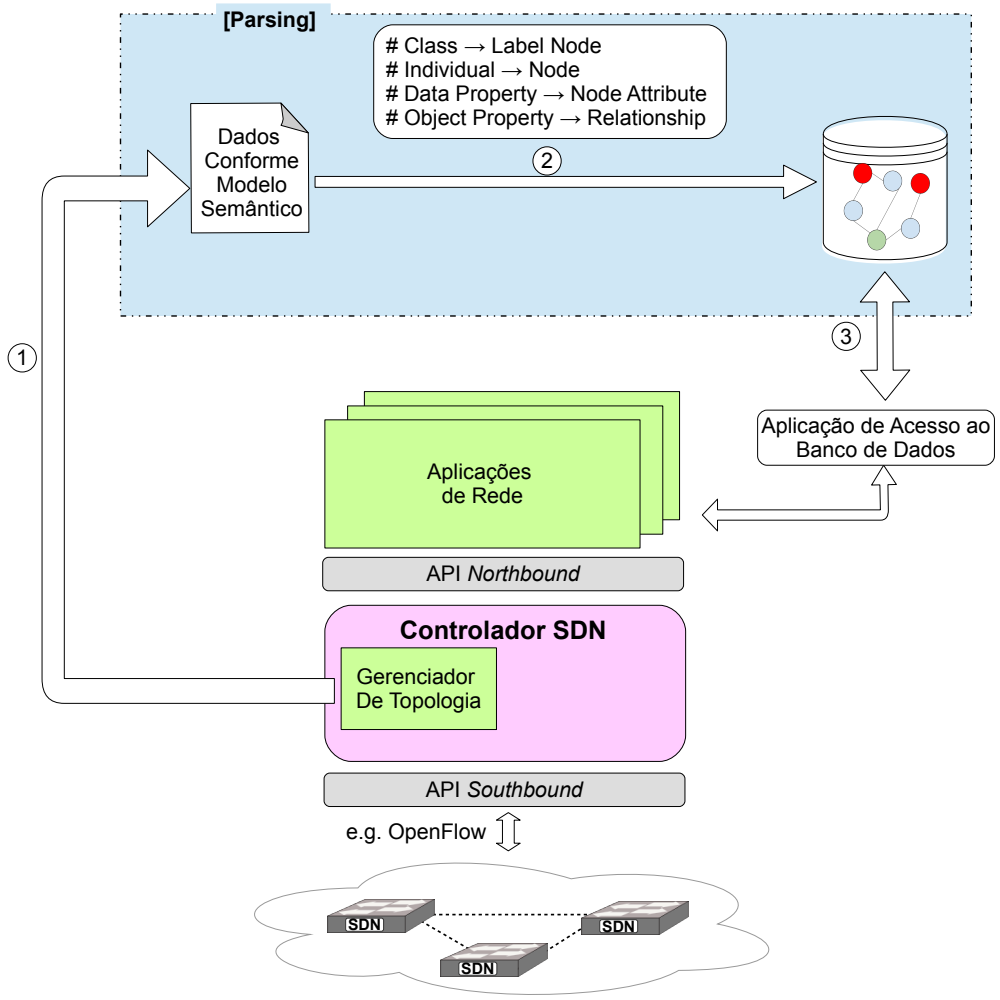

Figura 9 – Indexação no Banco de Dados com Suporte à Modelo Semântico - *Workflow* do *Parsing*

Para a tarefa de importação de dados e modelagem semântica, as seguintes regras foram aplicadas:

- As classes utilizadas foram: *Node*, *Port* e *Link*;
- Para cada nó da topologia foram geradas duas portas (*in* e *out*);
- Para cada conexão foram criados relacionamentos 2 *links* com entre as portas *in out*, uma vez que as conexões são unidirecionais.

O Pseudocódigo 3.1 descreve os procedimentos realizados para o *parsing* dos dados da topologia para o modelo semântico. As regras aplicadas estão de acordo com os conceitos apresentados no Capítulo 2. A entrada desse algoritmo deve ser um arquivo com os identificadores dos nós e suas conexões, e o arquivo de saída será no formato OWL RDF/XML. O gerador de topologias utilizado na avaliação experimental não descreve portas, como será apresentado na Seçã 3.6, portanto, o pseudocódigo faz a criação de duas portas para cada nó, entrada e saída, automaticamente.

Pseudocódigo 3.1 Geração do Modelo Semântico

enquanto !Fim do documento **faça**
  leia atual;
  se nó **então**
    crie um *Individual* da classe *Node*, *URI* ← identificador ;
    crie um *DatatypeProperty* do tipo *name*;
    crie um *Individual* da classe *Port*, *URI* ← identificador + "*_in*";
    crie um *Individual* da classe *Port*, *URI* ← identificador + "*_out*";
    crie um *ObjectProperty* do tipo *hasInboundPort* entre o *Node* e a porta de entrada;
    crie um *ObjectProperty* do tipo *hasOutboundPort* entre o *Node* e a porta de saída;
  senão
    crie um *Individual* da classe *Link*, *URI* ← identificadorNó1+"_"+identificadorNó2;
    crie um *ObjectProperty* do tipo *isSource* entre o *Link* e a nó1*_out*;
    crie um *ObjectProperty* do tipo *isSink* entre o *Link* e a a nó2*_in*;
    crie um *Individual* da classe *Link*, *URI* ← identificadorNó2+"_"+identificadorNó1;
    crie um *ObjectProperty* do tipo *isSource* entre o *Link* e a nó2*_out*;
    crie um *ObjectProperty* do tipo *isSink* entre o *Link* e a a nó1*_in*;
  fim se
fim enquanto

Como exemplo, trechos do arquivo de saída (`topology.rdf`), serão apresentados adiante. O Código 3.1 apresenta o nó "0" e suas portas de entrada e saída:

Código 3.1 – Trecho do arquivo topology.rdf, nó "0" e suas portas

```
1 <nml:Node rdf:about="urn:ogf:network:experiment.mt:2015:node0">
2   <nml:name>0</nml:name>
3   <nml:hasInboundPort rdf:resource="urn:ogf:network:experiment.mt:2015:0_in" />
4    <nml:hasOutboundPort
        rdf:resource="urn:ogf:network:experiment.mt:2015:0_out" />
5 </nml:Node>
6 <nml:Port rdf:about="urn:ogf:network:experiment.mt:2015:0_in">
7 </nml:Port>
8 <nml:Port rdf:about="urn:ogf:network:experiment.mt:2015:0_out">
9 </nml:Port>
```

Conforme mencionado anteriormente, o NML descreve conexões unidirecionais, dessa forma, a representação da conexão entre os nós "0" e "3" seria feita na forma "0_3" e

"3_0". O Código 3.2 exemplifica esses *Links*:

Código 3.2 – Trecho do arquivo topology.rdf, conexões entre os nós "0" e "3"

```
<nml:Link rdf:about="urn:ogf:network:experiment.mt:2015:0_3" />
<nml:Link rdf:about="urn:ogf:network:experiment.mt:2015:3_0" />
```

Após criadas as conexões, são acrescentadas às portas os *Object Properties isSink* e *isSource*, Código 3.3 apresenta o resultado:

Código 3.3 – Trecho do arquivo topology.rdf

```
<nml:Port rdf:about="urn:ogf:network:experiment.mt:2015:0_in">
   <nml:isSink rdf:resource="urn:ogf:network:experiment.mt:2015:3_0" />
</nml:Port>

<nml:Port rdf:about="urn:ogf:network:experiment.mt:2015:0_out">
  <nml:isSource rdf:resource="urn:ogf:network:experiment.mt:2015:0_3" />
</nml:Port>

<nml:Port rdf:about="urn:ogf:network:experiment.mt:2015:3_in">
   <nml:isSink rdf:resource="urn:ogf:network:experiment.mt:2015:0_3" />
</nml:Port>

 <nml:Port rdf:about="urn:ogf:network:experiment.mt:2015:3_out">
  <nml:isSource rdf:resource="urn:ogf:network:experiment.mt:2015:3_0" />
</nml:Port>
```

Para a implementação desses algoritmos é possível utilizar um *framework*, por exemplo, o Apache Jena (FOUNDATION, 2016), *open-source* e baseado na linguagem de programação Java.

3.4.2 Geração e Inserção do Grafo no GDB

Nessa etapa, os dados do modelo semântico são indexados no GDB Neo4j e conforme as seguintes regras:

- cada classe do NML é representada por um *label* no Neo4j, ou seja: *Node*, *Port* ou *Link*.
- cada indivíduo é representado por um nó e seu *label* é a sua classe.
- cada *DataType property* é representado por um atributo do nó.
- os *Object Properties* são representados por relacionamentos. O nome do relacionamento é dado pelo nome do *Object Property*.

O Pseudocódigo 3.2 descreve os procedimentos para a inserção do grafo no GDB. O arquivo de entrada desse algoritmo é o modelo semântico gerado pela primeira etapa do *workflow*. Para a implementação do algoritmo, o Neo4j possui uma API na linguagem Java, com funções que automatizam a tarefa de inserção, combinado com o *framework* Jena para processar o modelo semântico.

Pseudocódigo 3.2 Inserção do Grafo no GDB

```
enquanto !Fim do documento faça
  leia atual;
  se Individual então
    crie um nó, label ← ClassName, id ← URI;
    para cada Datatype Properties faça
      atributo ← Datatype Property, valor ← Value;
    fim para
    para cada Object Properties faça
      crie um relacionamento, tipo ← Object Property, com Individual;
    fim para
  fim se
fim enquanto
```

3.4.3 Consultas e Atualizações

Esta etapa é outra importante contribuição deste trabalho, uma vez que as informações da topologia estão disponíveis para consumo do controlador e aplicações SDN. O maior valor da proposta está em garantir que os dados, agora representados utilizando um modelo semântico, possam ser rapidamente processados, consultados e atualizados, de maneira que auxilie a tomada de decisões do controlador. Assim, com os dados indexados no GDB, as primitivas são implementadas em linguagem de consulta do banco de dados. As primitivas são respostas às consultas realizadas por aplicações do controlador SDN, por meio de interfaces *northbound*.

As primitivas previamente apresentadas foram reproduzidas na linguagem nativa do Neo4j, Cypher e estão descritas no Anexo A.

3.4.4 GDB para Modelo Semântico

Uma etapa que não está explícita na Figura 9 é a passagem dos dados do GDB para o modelo utilizado no *parsing*. Para esses processos, os dados já estão devidamente indexados no GDB e pretendem-se então extrair o modelo semântico, em um arquivo de saída. Essa etapa é importante, dado que as atualizações mais dinâmicas da topologia podem ser feitas diretamente no banco, com o objetivo de evitar um gargalo. Dessa forma, o caminho inverso manterá o modelo semântico atualizado com *snapshots* da rede. Além disso, um aspecto importante na utilização de modelos semânticos está na possibilidade do uso de raciocinadores (*reasoners*) para realizar inferências e encontrar inconsistências na ontologia. Essa característica não foi

explorada nesta proposta, mas possibilita trabalhos futuros. Assim sendo, as regras aplicadas são:

- cada nó é representado por um indivíduo e sua classe é definida pelo *label* do nó.
- cada atributo do nó é representado por uma *data property*.
- cada relacionamento é representado por uma *object property*.

O Pseudocódigo 3.3 apresenta as regras em formato de algoritmo.

Pseudocódigo 3.3 Geração do modelo a partir do GDB

```
enquanto !Fim do documento faça
  leia atual;
  se nó então
    crie um Individual, Class ← label, URI ← id ;
    para cada Atributos faça
      crie uma data property, Value ← valor;
    fim para
    para cada Relacionamentos faça
      crie uma object property, Individual ← nóRelacionado;
    fim para
  fim se
fim enquanto
```

3.5 Proposta para suporte de roteamento *Internet Protocol*(IP)

Conforme visto na Seção 3.3, o modelo semântico NML possibilitou a descrição da topologia do ponto de vista de conectividade. Entretanto, uma limitação foi identificada, que o modelo não prevê o atributo de custo de uma conexão, necessário para as primitivas de `setEdgeWeight` e `getEdgeWeight`. Além disso, uma possibilidade a ser explorada, que o modelo não abrange, é representação de dados da camada 3 do modelo *Open Systems Interconnection* (OSI), principalmente protocolo IP, utilizado na Internet.

Uma vez que o NML foi definido utilizando padrões da Web Semântica, que prezam pelo reuso e extensão dos modelos semânticos, foi realizado um estudo de adequação do NML que considere os dados de roteamento. O objetivo de modelar tais dados é padronizar e facilitar o compartilhamento destes entre controladores e aplicações SDN. Na próxima seção a proposta de extensão é detalhada.

3.5.1 Proposta

O protocolo IP é utilizado na camada de rede e sua principal função é o encaminhamento de pacotes de uma origem até o seu destino. Para isso são utilizados endereços

(chamados endereços IP) que são agregados em rotas, que definem o caminho que os pacotes devem ser encaminhados pelos roteadores em uma rede. Uma rota basicamente é definida por prefixo de destino e endereço de próximo salto (*next-hop*) podendo ter um custo atrelado. Um roteador possui uma tabela de roteamento, que é composta por uma coleção de rotas as quais são consultadas no momento do encaminhamento de um pacote.

Para modelar tais dados, utilizando como base o NML, foi criada uma subclasse de *Label* entitulada *IPAddress*, que por sua vez possui duas subclasses: *IPv4Address* e *IPv6Address*. Ambas possuem uma *data property* necessária, chamada *hasIPv4Address* e *hasIPv6Address*, respectivamente. Essas propriedades possuem como domínio a classe em questão a tem como *range* uma *string* que respeita uma expressão regular que valida somente valores que representem corretamente um endereço IPv4 e IPv6 (respectivamente). Outra *Data Property* criada é *hasCost*, que possui como domínio uma classe do tipo *Link* (já existente na ontologia do NML) e possui como *range* um valor do tipo *float*, podendo assim ser expressado um custo para um *link*.

No caso das *abject properties*, foram criadas *hasNextHopIPv4*, *hasDstIPv4*, *hasNextHopIPv6*, *hasDstIPv6*. Essas são usadas para criar uma relação que expresse uma rota. A classe *Route* deve ter uma ligação do tipo *hasDstIPv4* com uma classe do tipo *IPv4Address* e uma ligação do tipo *hasNextHopIPv4* também com uma classe do tipo *IPv4Address* (o mesmo vale para as versões para IPv6). Foi criada uma subclasse de *Service* chamada *RoutingService* que possui relacionamento com a classe *RoutingTable* por meio da propriedade *hasRoutingTable*. A classe *RoutingTable*, subclasse da *Network Object*, por sua vez se liga à classe *Route* por meio da relação *hasRoute*. Na Tabela 3 e na Tabela 4 são apresentados as *object properties* e as *data properties* propostas.

Nome	**Domínio**	***Range***
`hasDstIPv4`	Route	IPv4Address
`hasNextHopIPv4`	Route	IPv4Address
`hasDstIPv6`	Route	IPv6Address
`hasNextHopIPv6`	Route	IPv6Address
`hasRoute`	RouteTable	Route
`hasRoutingTable`	RoutingService	RoutingTable

Tabela 3 – *Object Properties* Propostas

Nome	**Domínio**	***Range***
`hasCost`	Link	*float*
`hasIPv4Address`	IPv4Address	*string* (expressão regular)
`hasIPv6Address`	IPv6Address	*string* (expressão regular)

Tabela 4 – *Data Properties* Propostas

Na Figura 10 é apresentado o diagrama de classes UML do *schema* do NML (HAM *et al.*, 2013a) juntamente com as classes propostas nesse capítulo.

Essa proposta é o resultado de um estudo inicial de extensão do NML. Para a validação da proposta, é necessário realizar testes com uso de uma ferramenta, e.g. Protégé [2] e verificar inconsistências. Essas análises serão feitas em trabalhos futuros.

3.6 Avaliação Experimental

Com o objetivo de realizar uma prova de conceito foi realizado uma avaliação experimental para analisar a viabilidade da proposta apresentada, considerando o tempo de resposta das consultas e a capacidade de recuperação de informação da topologia. As próximas subseções detalham o ambiente e os testes realizados.

3.6.1 Ambiente de Testes

Todos os testes da avaliação experimental foram executados em uma máquina de processador Intel i7 de 2.4 GHz, 8 Gigabytes de memória RAM. A versão do Neo4j utilizada foi a *Comunity Edition* 2.0.4, com licença GPLv3. Para a criação das topologias no banco de dados e execução de consultas foi utilizada a API Java do Neo4j. Apesar da API oferecer métodos pré-definidos de manipulação no banco, como a modelagem dos nós são segundo o modelo NML,

[2] Protégé - http://protege.stanford.edu/ (acessado em 29/02/2016)

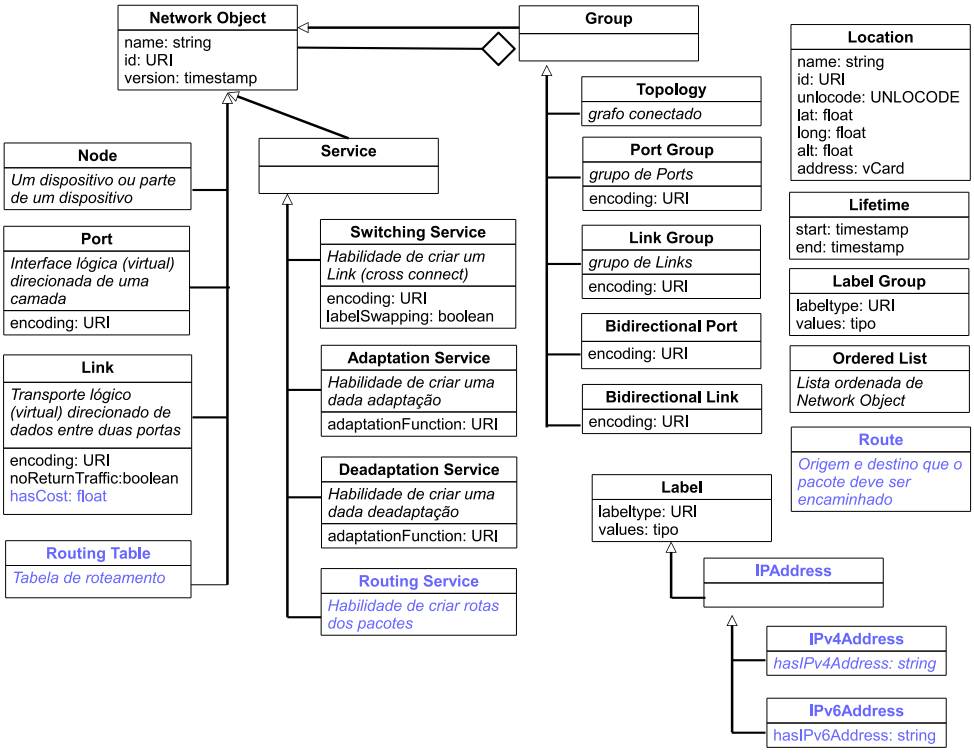

Figura 10 – Diagrama de classes do *schema* NML + classes e atributos propostos na extensão, marcados em azul (Adaptado de (HAM *et al.*, 2013a)

os métodos requerem adaptação, dessa forma, foi escolhida a criação e execução de consultas por meio da linguagem *Cypher*.

3.6.2 Gerador de Topologias

Conforme apresentado no Capítulo 1, uma das questões do trabalho apresentado nesta dissertação é a tarefa de gerenciamento de topologias. Nesse sentido, foram criadas quatro topologias de tamanhos diferentes utilizando o gerador de topologias BRITE (MEDINA *et al.*, 2001). Esse gerador possui algumas opções de modelos e vários parâmetros em cada modelo. Para esse caso de uso, a classe escolhida para a geração das topologias foi a *Flat Router-Level* e o modelo *RouterBarabasiAlbert*. Esse modelo gera uma topologia aleatória por meio do modelo proposto por Barabási e Albert (MEDINA *et al.*, 2001), o qual conecta os nós com uma conectividade preferencial usando uma probabilidade para gerar uma rede *power-law*, com alguns nós mais conectados. Outra característica desse modelo é que a criação de todos os nós no plano é feita da mesma forma e após a criação da topologia os atributos de largura de banda de todas as conexões também são definidos igualmente (MEDINA *et al.*, 2001). Para a geração das topologias, nesse caso, a localização dos nós no plano foi feita de forma aleatória.

O BRITE gera um arquivo com os nós da rede, seus atributos e as conexões entre eles. Para cada nó o gerador define sete atributos e para cada conexão nove atributos. Os atributos de cada nó são:

- *NodeId*: identificador único do Nó;
- *xpos*: coordenada x no plano;
- *ypos*: coordenada y no plano;
- *indegree*: grau de entrada do nó;
- *outdegree*: grau de saída do nó;
- *ASid*: identificador do AS;
- *type*: tipo (e.g. router, AS).

Já os atributos de cada conexão são:

- *EdgeId*: identificador único da aresta;
- *from*: nó de origem;
- *to*: nó de destino;
- *length*: distância Euclideana;

- *delay*: atraso de propagação;
- *bandwidth*: largura de banda (assinado pelo método AssignBW);
- *ASfrom*: caso topologia hierárquica, id do AS do nó de origem;
- *ASto*: caso topologia hierárquica, id do AS do nó de destino;
- *type*: tipo assinado da aresta pela rotina de classificação.

Entretanto, para a realização dos testes, apenas os atributos de identificador único do nó, *NodeId*, e nó de origem e nó de destino da conexão, *from* e *to* foram considerados.

O principal objetivo desse caso de uso é avaliar o comportamento do GDB em relação às primitivas, em tempo de resposta, utilizando diferentes tamanhos de topologias. Dessa forma, o foco está na tarefa 3 do *workflow* apresentado na Figura 9.

3.6.3 Aplicação de teste

Para avaliar o desempenho do GDB, nesse caso de uso, foram criadas 4 topologias de 10 (*tiny*), 100 (*small*), 1.000 (*medium*) e 10.000 (*large*) nós gerados pelo BRITE. Esses arquivos gerados com as topologias, foram pré-processados e uma aplicação, em Java, indexou os nós e suas conexões no Neo4j. Entretanto, para respeitar o modelo semântico, os dados foram criados no banco conforme modelagem apresentada na Seção 3.2. Para exemplificar a modelagem dos dados, a Figura 11 apresenta as conexões entre os nós “9"e “0", suas *Ports*, *Links* e relacionamentos. Como formalismo, as *Ports* de entrada e saída, foram identificadas, respectivamente, com seu id_{in} e id_{out}. Dessa forma, os grafos resultantes foram:

- 76 nós (160 relacionamentos) para a topologia *tiny*;
- 640 nós (1.760 relacionamentos) para a topologia *small*;
- 4.978 nós (11.912 relacionamentos) para a topologia *medium*;
- 109.932 nós (359.728 relacionamentos) para a *large*.

3.6.4 Análise de Resultados

Para realizar as métricas das consultas, a aplicação desenvolvida foi executada, individualmente e sequencialmente, as primitivas (consultas) com parâmetros aleatórios. Durante a realização dos experimentos foi observado que os primeiros tempos de cada primitiva foram muito superiores à média das demais consultas. Esse comportamento foi também observado no trabalho de Soundararajan e Kakaraddi (SOUNDARARAJAN; KAKARADDI, 2014). A explicação para tal fato é que na primeira execução o Neo4j coloca o grafo em cache e as demais

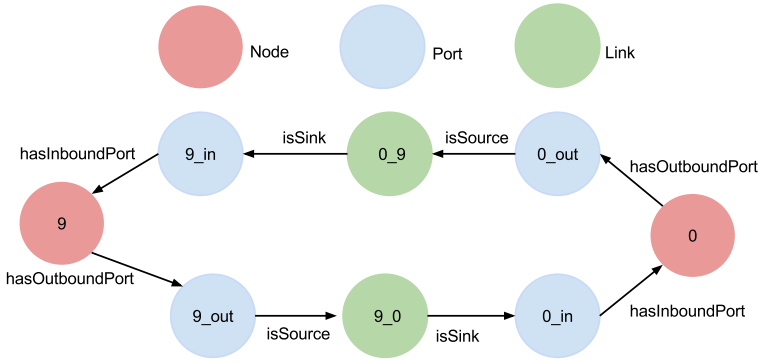

Figura 11 – Exemplo de relacionamento entre os nós “9"e “0"

execuções são realizadas no grafo em memória, o que economiza tempo. Por esse motivo, todos os primeiros resultados foram desconsiderados e cada primitiva foi executada 1.001 vezes em cada topologia. Além disso, os *outliers* não foram computados, tais valores foram considerados anormais pois não apresentaram padrão ou relação com os parâmetros.

Nas Tabelas 5 e 6 são apresentados a média, desvio padrão, 99 percentil e 1 percentil de cada primitiva, nas topologias *small* e *large*. A coluna de 1º valor se refere ao tempo desconsiderado para cálculo das estatísticas. Na Tabela 5 a primitiva `delete` não foi executada 1001 vezes, devido ao seu tamanho, ela foi executada 100 vezes.

Primitiva	Média	Desvio Padrão	Percentil 99
`setEdgeWeight`	8,78	3,23	23,02
`getEdgeWeight`	1,73	0,76	3,00
`countInDegree`	17,94	11,36	65,01
`countOutDegree`	8,35	3,46	23,00
`countNeighbors`	4,46	3,70	12,04
`doesRouteExist`	6,55	3,82	15,02
`computeMST`	1,12	0,66	2,00
`computeSSSP`	1,34	1,38	4,00
`computeKSSSP`	2,84	2,56	12,00
`computeAPSP`	1,04	0,84	4,01
`delete`	20,71	7,20	48,01
`insert`	3,66	3,26	15,02

Tabela 5 – Tempo (ms) de execução das primitivas - Topologia *small*

As primitivas com maior latência são `countInDegree`, `countOutDegree` e `delete`, comportamento verificado em todas as topologias. Para as operações de contagem de conexões de entrada e saída de um nó, esse resultado pode ser explicado devido ao número de *hops*, ou seja, os tipos diferentes de relacionamentos que é preciso percorrer para fazer a contagem (SOUNDARARAJAN; KAKARADDI, 2014). Por exemplo, para a contagem de conexões de entrada de um *Node* A, deve ser percorrido o seguinte padrão:

$$NodeA \leftarrow hasInboundPort \leftarrow Port \leftarrow isSink \leftarrow Link$$

Ou seja, a busca é feita a partir do *Node* A relacionado com sua(s) porta(s) de entrada, que por sua vez relacionada(s) como destino (entrada) de um *Link*. As setas representam

Primitiva	Média	Desvio Padrão	Percentil 99
`setEdgeWeight`	161,33	9,46	205,01
`getEdgeWeight`	1,70	0,74	4,00
`countInDegree`	854,53	146,77	1399,05
`countOutDegree`	425,17	68,36	699,02
`countNeighbors`	4,45	2,27	10,01
`doesRouteExist`	37,51	29,09	73,06
`computeMST`	1,44	1,25	3,02
`computeSSSP`	5,14	4,98	29,00
`computeKSSSP`	24,50	20,05	71,16
`computeAPSP`	1,04	0,68	3,01
`delete`	1053,89	162,55	1637,02
`insert`	3,57	3,21	16,01

Tabela 6 – Tempo (ms) de execução das primitivas - Topologia *large*

as direções do relacionamento, nesse caso, no sentido de entrada do nó. A contagem de entrada é dada pelo total *Links* do padrão apresentado.

O mesmo ocorre com a operação de exclusão, pois para essa primitiva, a consulta exclui o *Node*, suas *Ports*, *Links* conectados às portas e os relacionamentos com outras *Port*. Nesse caso, o número de *hops* é maior que nas operações de contagem de entrada e saída. Dessa forma, a latência está ligada ao nível de conectividade do nó excluído. Em seguida está a `setEdgeWeight` que no tempo geral das primitivas teve uma latência maior que as demais operações de leitura, ainda que na topologia *small* a média não tenha ficado alta. Como comparado no trabalho de Jouili e Vansteenberghe (JOUILI; VANSTEENBERGHE, 2013) o Neo4j apresenta um melhor desempenho em operações de somente leitura do que em operações de leitura-escrita. É possível validar ainda mais esse comportamento, comparando com os resultados da primitiva `getEdgeWeight`.

Para as demais primitivas, a Figura 12 mostra um comparativo entre as primitivas de leitura `computeSSSP`, `computeKSSSP` e `doesRouteExist` e a primitiva de escrita `insert` para as topologias *small*, *medium*, *large*. Não foram inseridas todas as primitivas devido aos altos valores, para manter uma escala que permita visualização dos resultados. Em geral, as primitivas de menores caminhos apresentam boa performance nas quatro topologias. Um comportamento interessante observado foi que a primitiva que calcula todos os pares de menor caminho (`computeAPSP`) é inferior quando comparado com a primitiva de *k* menores caminhos entre dois nós (`computeKSSSP`) e a de menores caminhos a partir de um nó específico (`computeSSSP`).

É possível deduzir que o GDB otimiza as consultas no caso de todos os pares de menores caminhos pois pode já calcular o menor caminho entre nós intermediários, o que não ocorre nas primitivas que consideram um nó ou dois específicos. Ainda nessa análise, a primitiva de `doesRouteExist` apresenta maior latência, uma vez que calcula o caminho entre nós específicos. A primitiva `computeMST` possui a mesma implementação de `computeSSSP`, como é possível observar nos seus resultados próximos.

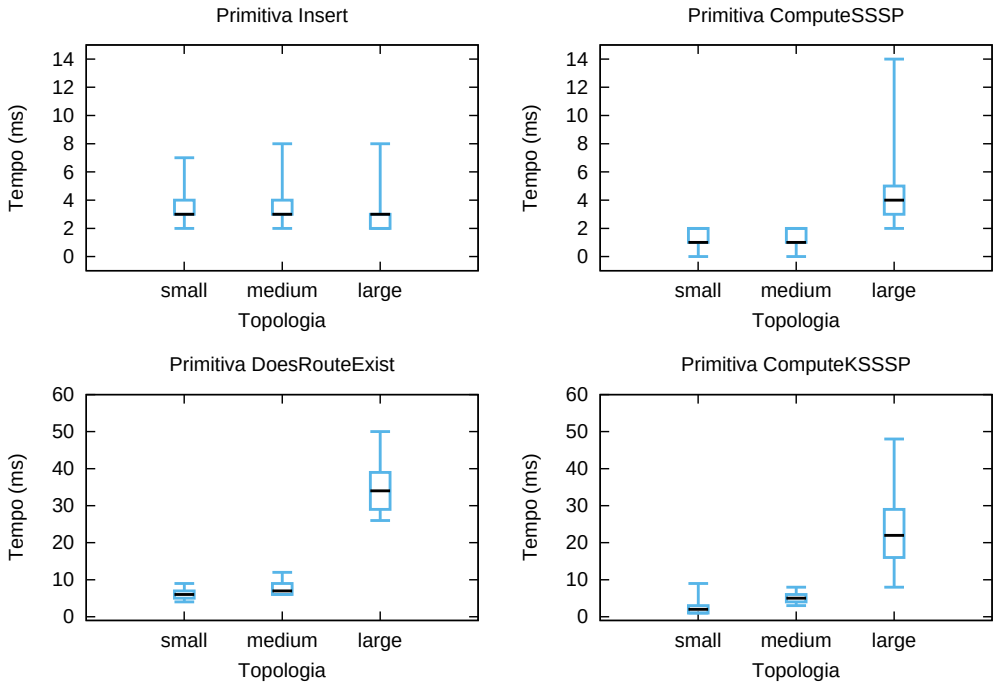

Figura 12 – Tempo de resposta das primitivas. Os gráficos *candlesticks* apresentam o valor médio, os quartis, e os valores max/min como 95-percentil.

3.6.5 Comparação com o Modelo Relacional

Com intuido de obter resultados de performance de um tradicional RDBM, foram reproduzidas um conjunto das primitivas SDN, da biblioteca NetGraph, usando uma aplicação Java com MySQL (version 5.6.17) (CORPORATION, 2015) e a linguagem de consulta SQL. Os resultados permitiram comparar as diferenças de implementação, bem como as vantagens e desvantagens em relação ao GDB. A Figura 13 apresenta o diagrama Modelo Entidade-Relacionamento (MER), um modelo lógico com a descrição das estruturas que foram armazenadas no banco. O modelo segue os relacionamentos do NML, como exemplificado na Figura 11, representando o grafo como tabelas usando chaves primárias e estrangeiras. Assim, os relacionamentos são de *um-para-muitos* entre *Node* e *Port*, enquanto que um *Link* é um relacionamento recursivo *muito-para-muitos* entre duas *Ports*.

Durante o projeto da aplicação de teste foi possível observar uma vantagem prática do GDB, em que tarefas de modelagem são consideravelmente mais naturais comparadas ao RDBM, o que é esperado, considerando que é um projeto de rede, com interconexão de dados. Isso pode ser confirmado com o MER, no qual os relacionamentos entre as entidades são simples e sem direções (ROBINSON *et al.*, 2013), ou seja, o modelo é claramente adaptado. Tabela 7 apresenta os tempos de execução para um subconjunto das primitivas no caso da topologia *large*.

As primitivas de leitura escolhidas foram: a primitiva de contagem de grau de entrada, `countInDegree`, duas primitivas de menores caminhos, a de todos os pares de menores

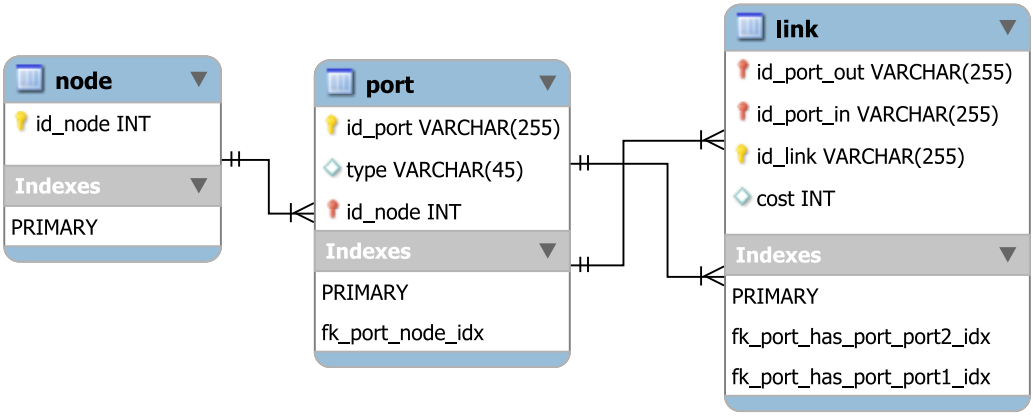

Figura 13 – Modelo Entidade-Relacionamento (MER)

caminhos `computeAPSP` e a de menor caminho de um nó para todos os outros `computeSSSP` e as primitivas de escrita de inserção e exclusão, `insert` e `delete`.

Os resultados para as primitivas `delete` e `countInDegree` são mais rápidas com o MySQL do que com o Neo4j, um fato que pode ser esperado a partir da latência acumulativa (número de *hops*) explicado na seção anterior. Em contra partida, no caso de inserção, o tempo de execução da primitiva `insert` foi muito mais alto do que no Neo4j, devido à necessidade de inserir dados em duas tabelas (*Node* e *Port*). Para a primitiva `computeSSSP` os resultados foram um pouco mais lentos que no Neo4j. Destaca-se aqui o fato de que para calcular as primitivas de menores caminhos uma implementação Java (baseada em Dijkstra (LITERATE-PROGRAMS, 2015)) foi necessária devido à falta de consultas do tipo de menores caminhos nativas no MySQL. Primeiramente, a implementação adaptada do algoritmo Dijkstra carrega em memória todos os nós e suas adjacências utilizando consultas do tipo `SELECT`. A média de tempo para essa tarefa de pré-execução dos dados é de aproximadamente 2 segundos.

Novamente, as primitivas orientadas a grafo nativas do Neo4j podem ser vistas como vantagens do GDB sobre o RDBM em uma perspectiva de desenvolvimento de aplicação. Do mesmo modo, a primitiva `computeAPSP` apresenta desempenho comparável, mas não requer qualquer esforço de desenvolvimento extra em uma abordagem GDB devido a funções nativas do Neo4j para cálculo de todos os pares de menores caminhos. Dessa forma, pode-se concluir que a utilização de um GDB para o cenário de redes é mais adequado do que um modelo tradicional, RDBM. As justificativas não ficam no mérito do tempo, uma vez que, para o cenário do RDBM os resultados de tempo poderiam ser otimizados, a partir de técnicas de

Primitiva	Média	Desvio Padrão	Percentil 99
`countInDegree`	1,39	4,57	22,02
`computeSSSP`	18,13	3,82	26,00
`computeAPSP`	2,11	1,39	7,00
`delete`	162,86	79,93	405,00
`insert`	137,36	43,48	300,00

Tabela 7 – Tempo (ms) de execução das primitivas no RDBM - Topologia *large*

desnormalização. A principal vantagem está na modelagem mais natural e na implementação de funções nativas do GDB.

3.6.6 Reproducibilidade de Experimentos

Todos os dados e códigos usados para os testes, apresentados nesta seção, estão disponíveis em um repositório público.[3] Os *datasets* incluem os modelos NML, topologias BRITE, códigos do *parsing*, consultas em *Cypher* do Neo4j, códigos da aplicação do MySQL e arquivos gnuplot [4] para geração de gráficos.

[3] NML-Neo4j - https://github.com/intrig-unicamp/NML-Neo4j (acessado em 25/04/2016)
[4] Gnuplot - http://www.gnuplot.info/ (acessado em 29/02/2016)

4 Cenários de Avaliação

Com o objetivo de validar a proposta de utilização de GDBs para o armazenamento e recuperação de informações de topologias, este capítulo apresenta dois casos de uso nos contextos de multidomínios SDN e virtualização recursiva. O foco dos cenários está na indexação dos dados da topologia e a reprodução de primitivas de consulta e atualização usando o GDB. Na avaliação, foi analisada a compatibilidade do grafo de propriedades com os cenários, portanto, os resultados apresentados no capítulo não fazem uso do modelo NML, que ficou para os trabalhos futuros.

4.1 Multidomínios SDN

O caso de uso explorado nesta seção considera o cenário de múltiplos domínios administrativos, descrito no ONF *Technical Recommendation* (TR) 502 (ONF-TR-502, 2014). Nesse cenário, múltiplos administradores possuem sua própria rede (subredes) e estão interligados. A Figura 14 apresenta um exemplo, cada cor (A - vermelho, B - verde e C - azul) representa um domínio e as conexões entre eles. Os retângulos representam elementos de rede, as linhas representam *links* que são estabelecidos por meio de portas externas.

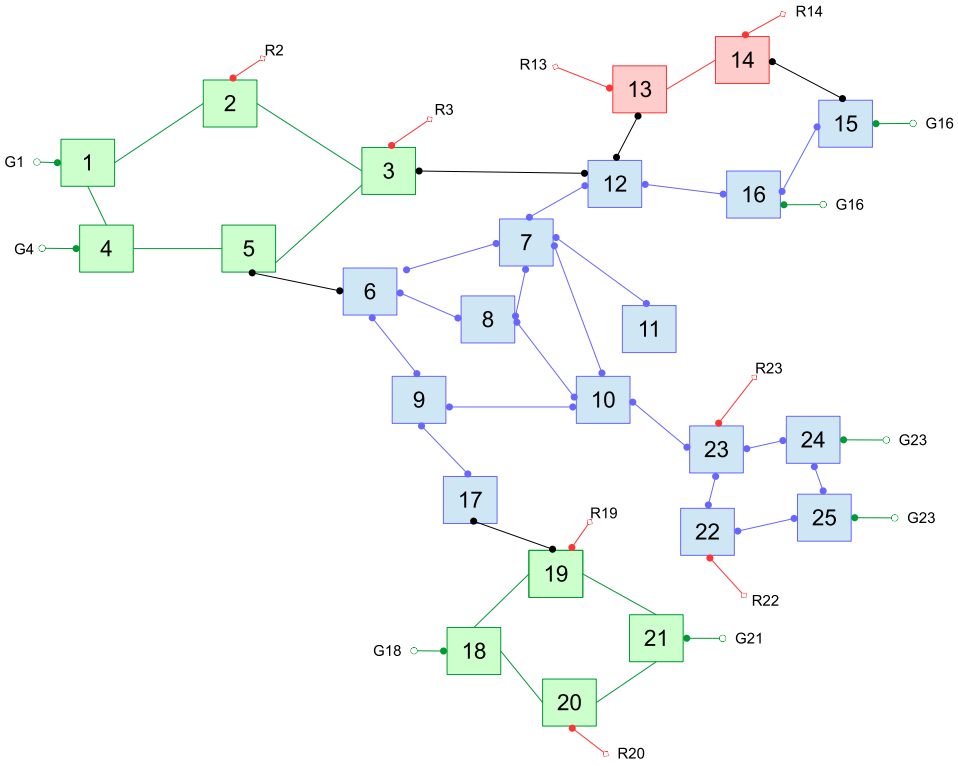

Figura 14 – Redes com múltiplos domínios (adaptado de (ONF-TR-502, 2014))

Considerando que cada domínio possua um controlador SDN, o mesmo terá a vi-

são completa dos elementos de rede de seu domínio, mas terá uma abstração simplificada dos demais domínios. Os controladores podem ter diferentes associações, como ilustrado na Figura 15. Na primeira situação, o controlador B (em verde) oferece um serviço que abrange o seu domínio e o domínio C (em azul). O controlador A (em vermelho) não possui relação de negócio com o controlador C mas haverá interação entre ele e os recursos do controlador C nos casos de virtualização fornecida pelo controlador B. Na segunda situação, os relacionamentos e virtualização estão em outra ordem, agora pelo controlador C. E por fim, na terceira situação, o controlador A tem uma relação contratual com os dois controladores, ou seja, ele tem visibilidade dos *links* e pode possuir certo nível de controle sobre os controladores B e C. Assim, o controlador A pode ou não otimizar o uso dos recursos dos controladores de acordo com critério de escolha, e.g. custo monetário, disponibilidade ou latência (ONF-TR-502, 2014).

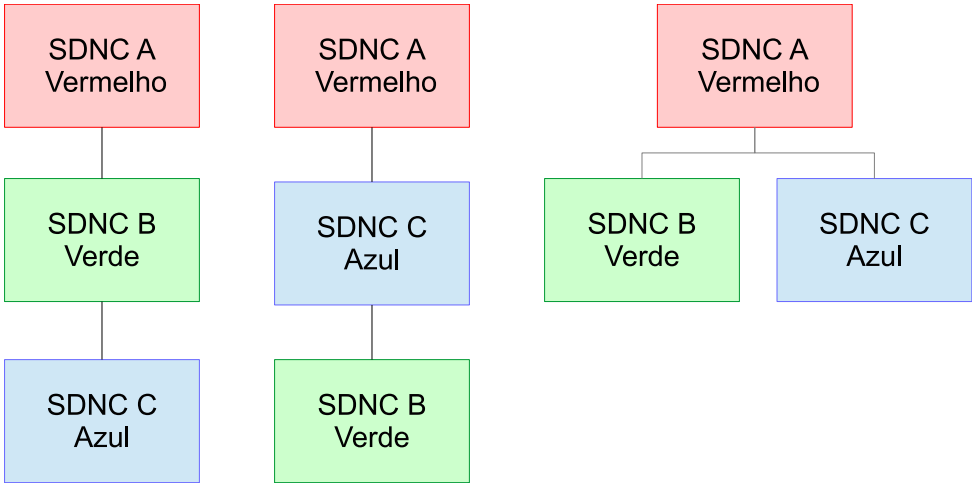

Figura 15 – Diferentes associações entre controladores (Adaptado de (ONF-TR-502, 2014))

Considerando múltiplos domínios, um outro cenário é o de coordenação C2C, isso significa que os controladores SDN são como *peers* sem a existência de orquestração de um controlador superior. A Figura 16 ilustra um exemplo de conjunto de domínios de rede (NCD). O controlador SDN branco atua como cliente e possui relacionamento direto com NCDs 1 e 2, mas com o NCD 4 um relacionamento indireto. Já o NDC 3 não possui um controlador SDN, nesse caso, é preciso existir um protocolo de roteamento ou um acordo previamente estabelecido. Os limites administrativos são também fronteiras de negócio, nesse caso, questões contratuais e segurança, ocultação de informação e confiança são indispensáveis (ONF-TR-502, 2014).

Exemplos de informações trocadas entre controladores são:

- adjacência do controlador e descoberta de capacidade;
- informação de estados e atributos;
- descoberta de topologia e vizinhos no plano de dados;

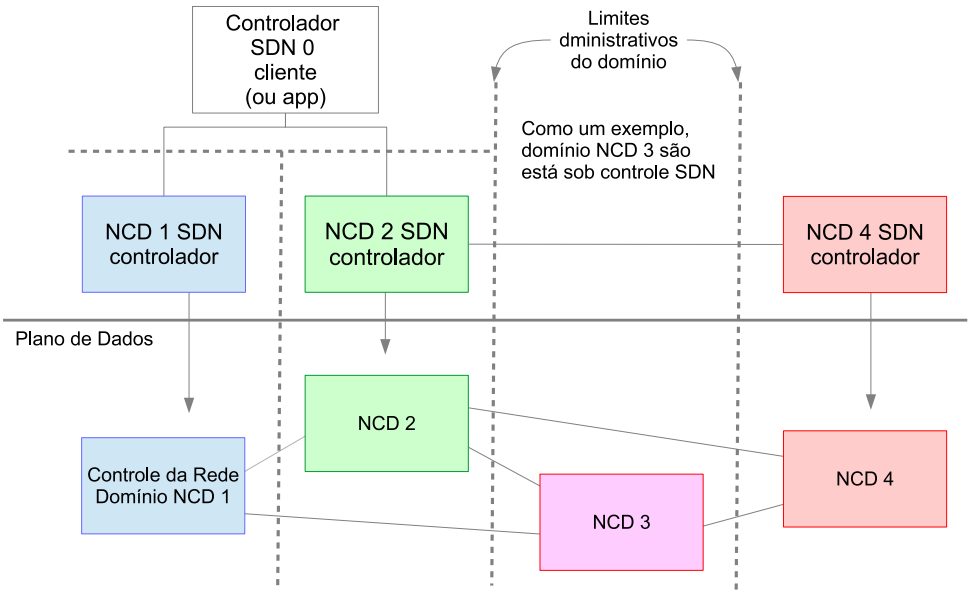

Figura 16 – Exemplo de coordenação C2C (Adaptado de (ONF-TR-502, 2014))

- informação de caminhos, e.g. custo e tamanho de rota.

Ressaltando que tais trocas são feitas dentro das políticas previamente estabelecidas no domínio.

4.1.1 Modelo de Informação

Conforme mencionado no início do capítulo para este caso de uso não será utilizado o modelo semântico, para garantir performance dos testes. Entretanto, para modelar os dados no banco, o modelo de informação utilizado é o ONF-CIM, descrito em ONF TR 512 (ONF-TR-512, 2015). O foco do ONF-CIM está na representação dos recursos do plano de dados, que será utilizado por um controlador SDN, que significa que os dados que dão suporte a informação de encaminhamento, que transforma uma rede de adjacência virtual. O ONF TR 513 (ONF-TR-513, 2015), apresenta as diretrizes para o desenvolvimento e uso do ONF-CIM. A proposta é permitir que o modelo seja derivado e refatorado para visões do modelo específicas, ou seja, selecionando diferentes subconjuntos de artefatos s(objetos, atributos, relacionamentos, etc.) a fim de atender interfaces específicas do domínio.

A modelo ONF-CIM está formalizado em UML e as principais classes consideradas nesse caso de uso são:

- *Network Control Domain/View* (NCD): representa o escopo de controle de um controlador SDN de uma rede específica.
- *Forwarding Domain* (FD): é a partição lógica para representar potencial para encaminhamento.
- *Logical Termination Point* (LTP): representam portas internas e na borda de um FD.

- *End Point* (EP): representa o acesso para encaminhamento e/ou adjacência, que deve ser associado a um LTP.

4.1.2 API de Transporte

A proposta *Transport* API do ONF, documentada em (ONF, 2015) explora o conceito de diferentes visões para cada cliente, principalmente controladores. A visão exportada dependerá da posição de um observador que invocou a *Transport* API, o qual em diferentes posições obterá diferentes perspectivas da(s) topologia(s). A Figura 17 apresenta a referência desse caso de uso. Considere um observador na posição 1, a visão obtida será de uma topologia de um nó (B) sem qualquer *link*. A posição 2, obterá uma visão de uma topologia com os nós A e C e o *link* entre eles (LTPs 3 e 4). Na posição 3, o API *Invocation Point* (API-IP) o observador obterá a visão de uma topologia sem elementos. E por fim, um na posição 4, a visão da topologia com os nós 1 ao 5 e todos os *links* entre eles.

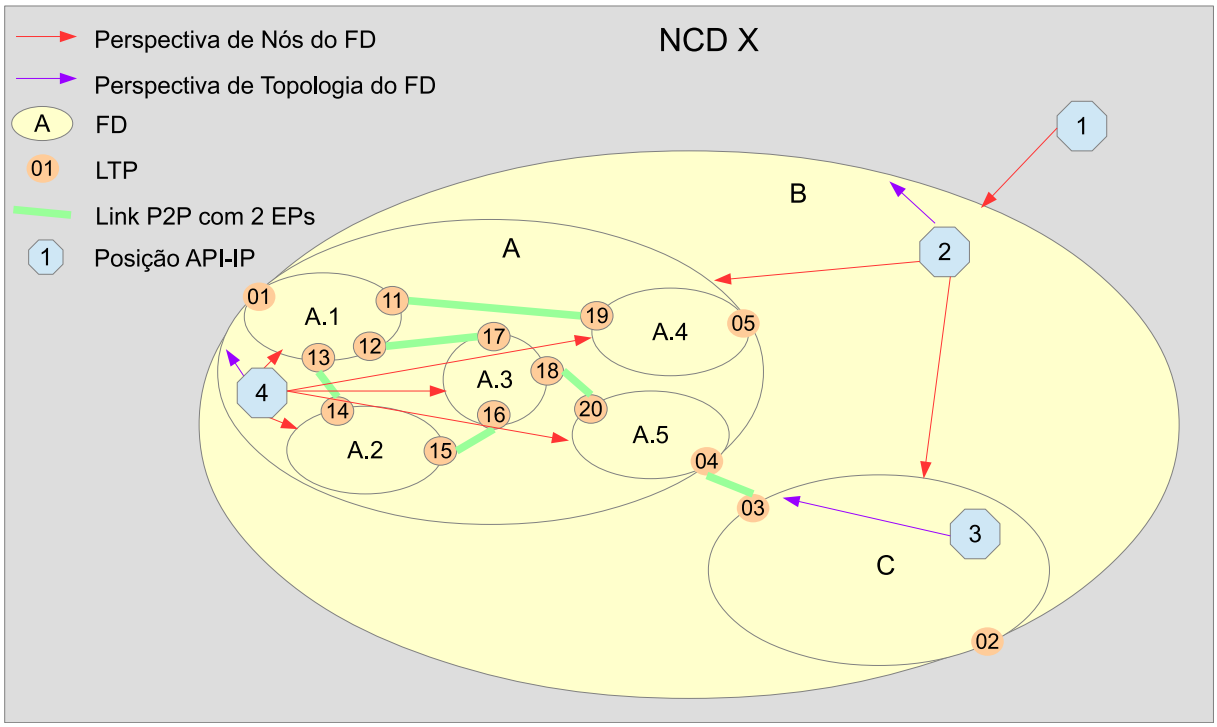

Figura 17 – Figura de Referência do projeto "*Transport* API" (ONF, 2015)

4.1.3 Avaliação Experimental

O objetivo dessa avaliação experimental foi verificar a compatibilidade do cenário com o GDB. Foram avaliados os seguintes aspectos:

1. Modelagem dos dados no GDB;
2. Modelagem das primitivas propostas;

O cenário de exemplo apresentado na Figura 19 foi considerado para avaliação. Esse exemplo apresenta o NDC do Controlador 1, a partir de visões exportadas pelos Controladores 2 e 3, ou seja, esse controlador conhece a topologia de cada outro controlador baseado no que é permitido visualizar, podendo existir políticas restritivas, por exemplo. É possível verificar a exportação de uma visão na Figura 18. Baseado em suas políticas, o Controlador 3, exporta a visão com apenas os LTPs de borda e a conexão com o domínio do Controlador 2 (ONF, 2015).

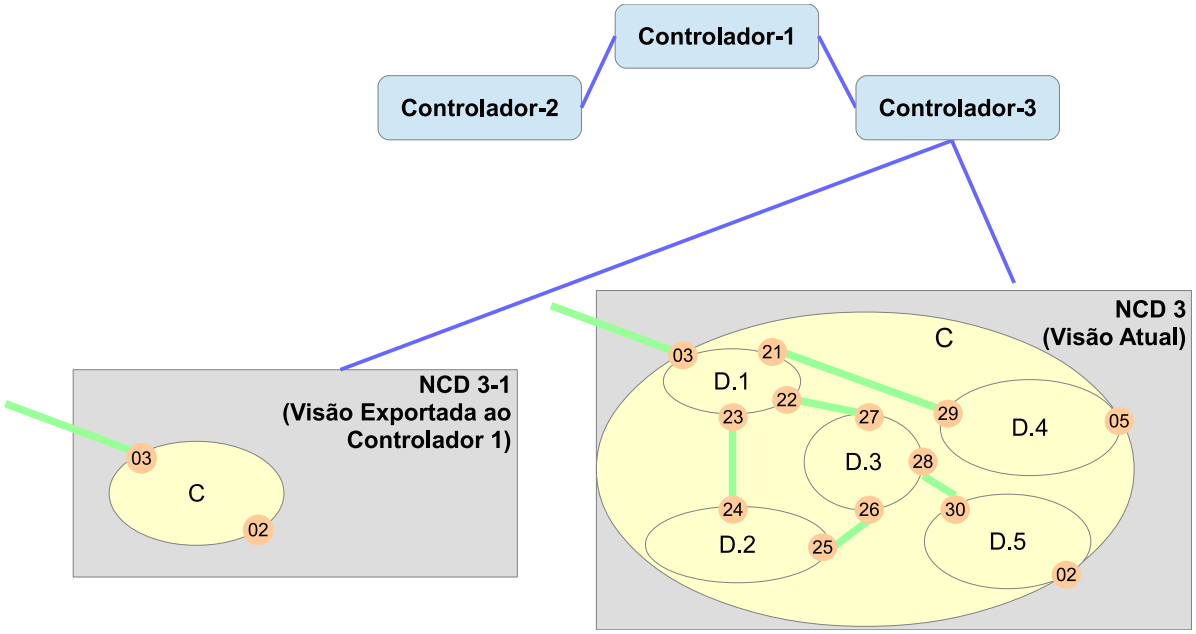

Figura 18 – Exemplo de exportação de visão (ONF, 2015)

Dado que o Controlador 1 possui uma visão montada a partir das visões importadas, uma aplicação pode consultar essas informações por meio de primitivas acessadas via *Topology* API:

- `GetTopologyEncompassedByFD(FD_ID)` realiza uma busca dos FDs que estão contidos em um FD;
- `GetServiceEndPointDetails(FD_ID)` busca quais são os SEP de um FD;
- `GetNodeDetails(FD_ID)` realiza uma busca de LTPs conectados a um FD;
- `GetLinkDetails(FD_IDorigem, FD_IDdestino)` retorna as portas de um *link* entre dois FDs.

Dessa forma, o principal objetivo foi indexar as visões de um controlador no Neo4j e modelar as novas primitivas em linguagem de consulta, Cypher. Conforme apresentado no Capítulo 2, o modelo de grafo do Neo4j, grafo de propriedades, permite atributos nos nós e relacionamentos. Portanto, para replicar o conceito de subgrafos, como o exemplo apresentado na Figura 19, no qual FD-A e FD-C estão contidos em FD-B, as possibilidade são: (*i*) por

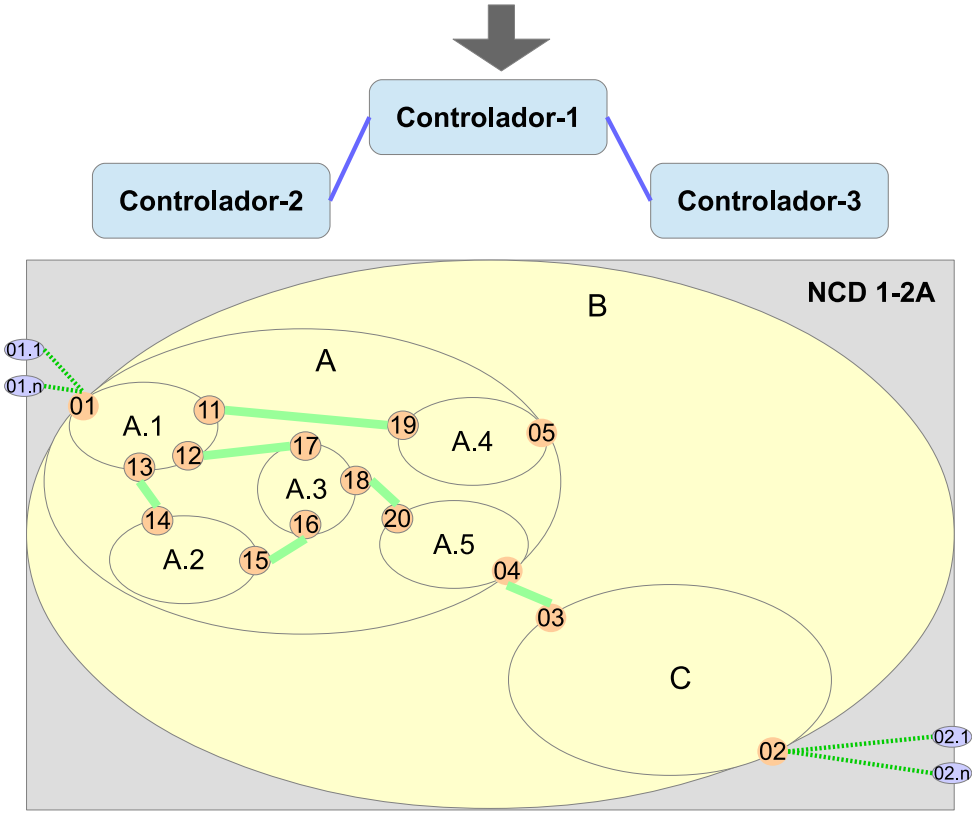

Figura 19 – Cenário de Exemplo do "*Transport* ONF-CIM"(ONF, 2015)

meio de consultas com restrições para buscar os subgrafos e executá-las quando necessário ou (*ii*) criar um nó para representar o subgrafo e relacioná-lo com os nós e relacionamentos que façam parte. A abordagem que melhor atende o cenário é a segunda, uma vez que um controlador receberá as visões dos demais controladores e construirá uma visão combinada às outras recebidas. A primeira abordagem, atenderia no caso do banco de dados esta indexado com todas os subgrafos completos, de todos os controladores, uma condição que dificilmente será atendida em cenário real. A modelagem dos dados no Neo4j é apresentada na Figura 20.

Nós do tipo FD se relacionam com nós do tipo LTP, por meio do relacionamento *hasPort*. Os nós do tipo LTP possuem relacionamentos do tipo *link* com outros nós do mesmo tipo e relacionamentos do tipo *hasService* com nós do tipo SEP. Por fim, a representação de FD contido em outro é feita por relacionamentos do tipo *encompass*. Essa modelagem permite ainda a definição de um atributo para o destino da conexão, relacionamento *link*. Por exemplo, um controlador exportar a visão de seus LTPs e *link* com um LTP de outro domínio, com esse atributo, o controlador que recebe a exportação, consegue completar sua visão no banco de dados. A Figura 21 apresenta a visualização dos dados do cenário já indexados no Neo4j. Uma vez que as visões foram indexadas no Neo4j, o próximo passo foi reproduzir as primitivas para o consumo de aplicações. As consultas foram escritas na linguagem Cypher e estão detalhadas no Anexo B.

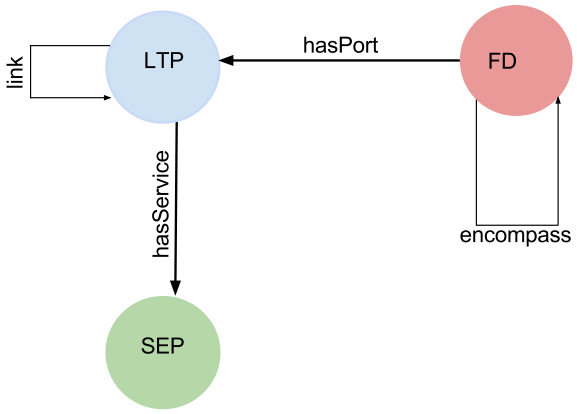

Figura 20 – Modelo lógico de multidomínios usado no banco de dados

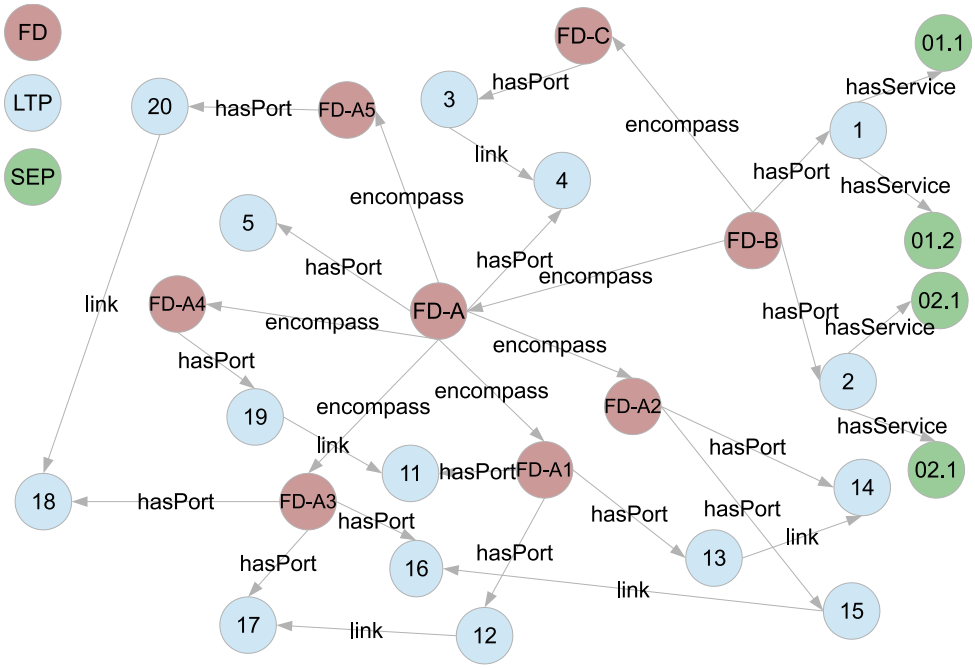

Figura 21 – Visão dos dados indexados no Neo4j

4.1.4 Análise dos Resultados

Não foram utilizadas métricas de tempo de execução para esta avaliação, visto que o capítulo anterior já apresentou resultados de desempenho do Neo4j. O foco desta avaliação está na compatibilidade do modelo com o GDB.

Foi possível reproduzir o cenário de subdomínios respeitando o modelo ONF-CIM. Além disso, as primitivas de aplicações também foram escritas na linguagem nativa Cypher. Do ponto de vista de implementação, as tarefas foram exequíveis de maneira simples, pois o grafo de propriedade do Neo4j atendeu as necessidades de modelagem. Para as primitivas reproduzidas na linguagem Cypher também não foi necessário o uso de recursos mais custosos da linguagem.

Os resultados foram satisfatórios e promissores, pois se mostra viável o uso de GDB em cenários para troca informações entre controladores multidomínios. Os dados são armazenados de maneira persistente, além do oferecimento de consultas e atualizações.

4.2 Virtualização Recursiva

O caso de uso explorado nesta seção considera o cenário de virtualização recursiva do projeto UNIFY (UNIFY, 2016), destinado a pesquisa, desenvolvimento e avaliação de medidas para orquestrar, verificar e observar a entrega de serviço das redes ponto-a-ponto por meio de agregação e núcleo de redes para data centers (MIJUMBI *et al.*, 2015). A API do UNIFY unifica em um ponto de referência os recursos de computação, armazenamento e rede, permitindo a virtualização da orquestração multicamadas do VNF-FG para o encadeamento rápido e flexível do serviço. A proposta do UNIFY é semelhante ao NFV entretanto recorre a arquitetura de controle do SDN (CARROZZO *et al.*, 2015). É um projeto cofundado pela EU FP7 e conta com a parceria de provedores de serviço, fornecedores, universidades e institutos de pesquisas.

A arquitetura do UNIFY explora o conceito de recursividade na orquestração, dado que a virtualização de computação, armazenamento e domínio de rede é multinível, para automatizar o provisionamento de recursos correspondentes há necessidade da programação recursiva. Sua arquitetura não limita a quantidade de domínios que podem ser colocadas juntas na hierarquia multinível (SZABO *et al.*, 2015).

Uma visão geral da arquitetura do UNIFY é apresentada na Figura 22. A arquitetura é composta por: *Service Layer* (SL), *Orchestration Layer* (OL) e *Infrastructure Layer* (IL), componentes de gerenciamento, *Network Functions System* (NFS) e os pontos de referência, conforme apresentado em (SZABO *et al.*, 2014).

Na camada de serviço (SL) estão o gerenciamento da virtualização e funções de negócio com o ciclo de vida dos serviços, na Figura 22 elas estão identificadas como "Gerenciamento de Serviço/ Funções de Adaptação". Para tais funções de gerenciamento estão *Element Management Systems* (EMS), os *Operational Support Systems* (OSS) e os *Business Support Systems* (BSS) dos provedores de serviços, *Service Provider* (SP). Para as funções relacionadas à virtualização, estão as funções de gerenciamento de ciclo de vida de serviços de rede virtualizadas, ciclo de vida de *Virtual Network Function* (VNF) e orquestração dos recursos presentes na camada mais baixa.

Os usuários finais/empresas são os consumidores dos serviços de telecomunicação oferecidos pelo SP, que os gerencia por meio de OSS ou BSS. Já os usuários UNIFY são os que consomem os Serviços de Recurso UNIFY diretamente. Os desenvolvedores podem desenvolver e testar as funções de rede ou pacotes de serviços.

A camada de orquestração (OL) possui dois principais componentes: (*i*) o *Resource*

Orchestration (RO), composto por *Virtualizers*, orquestração de recursos entre eles e os recursos adjacentes e executores de políticas e (*ii*) o *Controller Adapter* (CA) composto por funções de abstração de recursos de domínio global e virtualização de diferentes tipos de recursos, tecnologias, fornecedores ou administrações.

A camada de infraestrutura (IL) é composta por recursos, agentes de recurso local e controladores. Nos recursos estão todas as opções de computação, armazenamento e rede. Juntamente com eles estão os agentes locais, e.g. OpenFlow *switch*. Nos controladores estão as funções de virtualização de um único domínio, e.g. controlador SDN.

No gerenciamento estão os componentes de gerenciamento de: infraestrutura, de *Network Function* (NF) e suas bases de informação relacionada e das camadas OL e IL. A *Network Function Information Base* (NF-IB) utilizada pelo RO, define as informações relacionadas à NF necessárias para o serviço agnóstico de orquestração.

No sistema de funções de rede (NFS) estão as NFs instanciadas, incluindo dados, controle e componentes do plano de gerenciamento e o encaminhamento correspondente.

Os pontos de referências são os identificadores dos relacionamentos ponto-a-ponto entre "produtor" e "consumidor" dos blocos funcionais. As informações trocadas entre eles são modeladas por um protocolo neutro de modelo de informação. A arquitetura do UNIFY não será mais aprofundada aqui, pois não é o foco desse caso de uso. A documentação completa da arquitetura está disponível em (SZABO *et al.*, 2014).

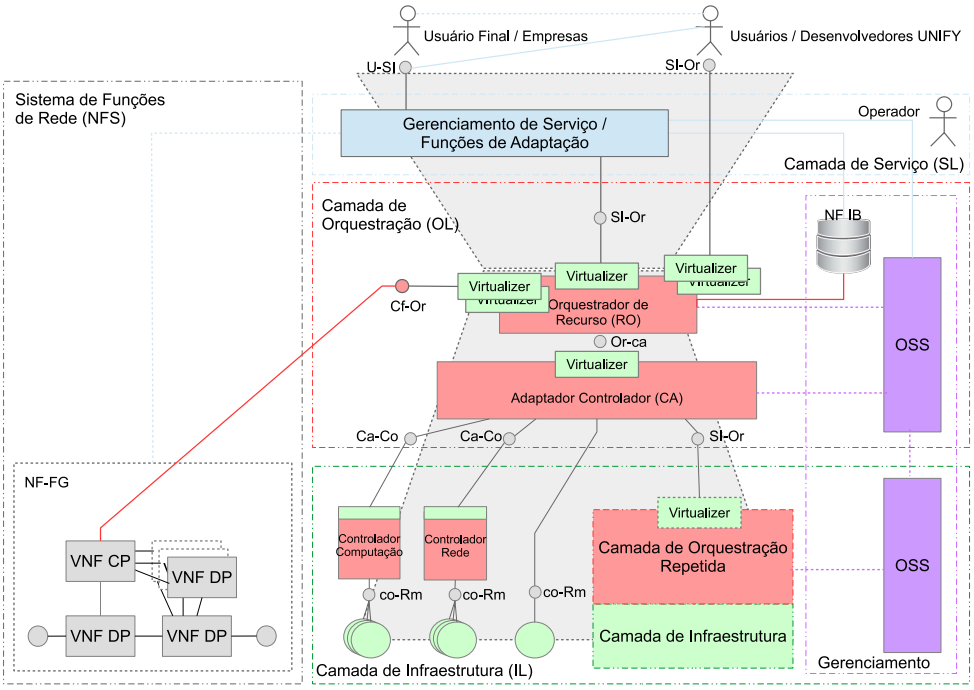

Figura 22 – Visão Geral da Arquitetura UNIFY (Adaptado de (SZABO *et al.*, 2014))

4.2.1 *Virtualizer*

O responsável pela alocação dos recursos abstratos e capacidades a um consumidor específico é o *Virtualizer*. Ou seja, recursos de computação, armazenamento, rede, ambientes de execução entre outros (SZABO *et al.*, 2014).

A junção de *software* e abstração (virtualização) de rede é definida como BiS-BiS (SZABO *et al.*, 2015). A Figura 23 apresenta um exemplo de virtualização de BiS-BiS. Um BiS-BiS é a virtualização de um *Forwarding Element* (FE) conectado a um *Compute Node* (CN), que é capaz de executar NFs e conectá-los ao FE.

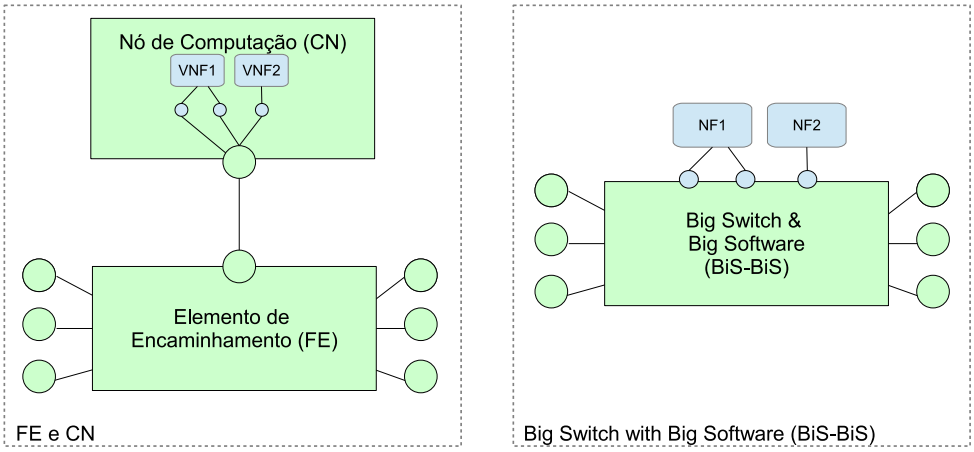

Figura 23 – Exemplo de Virtualização de BiS-BiS (SZABO *et al.*, 2014)

4.2.2 Modelo de Dados

Assim como o caso de uso apresentado na seção anterior, o cenário explorado neste caso de uso não fará uso do modelo semântico e sim do modelo de dados utilizado pelo UNIFY *Virtualizer* é o YANG, que utiliza o formato JSON (SKöLDSTRöM *et al.*, 2015). Entretanto, para a realização da avaliação experimental, o modelo não foi importado para simplificar os experimentos, pois o foco do caso de uso foi explorar a compatibilidade do banco de dados baseado em grafo com o cenário. Para a modelagem no banco, as entidades foram consideradas de acordo com o cenário, ou seja, as entidades comuns ao BiS-BiS.

4.2.3 Avaliação Experimental

O objetivo dessa avaliação experimental foi implementar a arquitetura proposta no contexto do projeto UNIFY e analisar sua compatibilidade com o GDB. Dessa forma, foram realizadas as seguintes tarefas:

- Modelagem dos dados do *Virtualizer* no GDB;
- Reprodução das primitivas propostas.

A Figura 24 apresenta a modelagem utilizada no Neo4j para armazenar os dados do *Virtualizer*, ou seja, os recursos de infraestrutura (InfraNode), funções de rede (NFs) e portas (*Ports*). Além disso, o modelo armazena também as agregações do BiS-BiS. As alocações das funções, VNF-FG, são representadas pelos relacionamentos de fluxo. Dessa forma, conforme modelagem, os nós do tipo InfraNode possuem relacionamentos *hasPort* com nós do tipo *Port* e relacionamentos *hasNF* com nós do tipo NF. Nós do tipo BisBis possuem relacionamentos *hasPort* com nós *Port* e *hasNode* com nós InfraNode. As conexões físicas ou lógicas entre InfraNodes é representada pelo relacionamento *hasLink* (entre as portas). A representação do VNF-FG dos serviços é feita pelos relacionamentos do tipo *hasFlow* entre os nós do tipo *Port*. O relacionamento *hasFlow* possui ainda os atributos `match` e `action`, que identificará os fluxos.

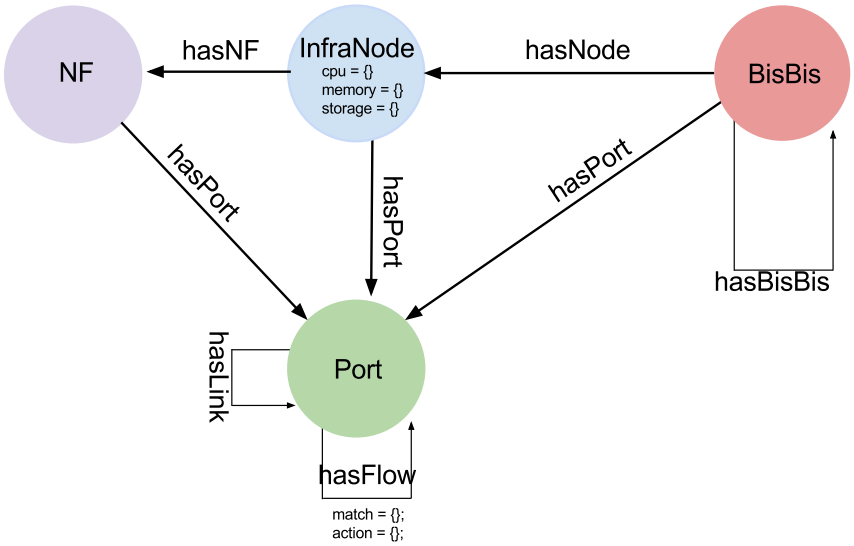

Figura 24 – Modelo lógico do UNIFY *Virtualizer* usado no banco de dados

A Figura 25 apresenta a visão de dados de exemplo indexados no Neo4j.

Após a indexação dos dados no Neo4j, as primitivas do consumo do RO, foram reproduzidas na linguagem Cypher:

- Conjunto de BiS-BiS;
- Conjunto de *InfraNodes* e suas NFs;
- *Links* entre portas e seus respectivos *InfraNodes*;
- Alocações por *tag*.

Além dessas primitivas propostas de leitura, uma primitiva de geração de BiS-BiS foi reproduzida, a partir das seguintes regras de agregação:(*i*) todos os recursos de infraestrutura e (*ii*) as portas com apenas um *link* (de entrada ou de saída). Todas as primitivas, em Cypher, estão detalhadas no Anexo D.

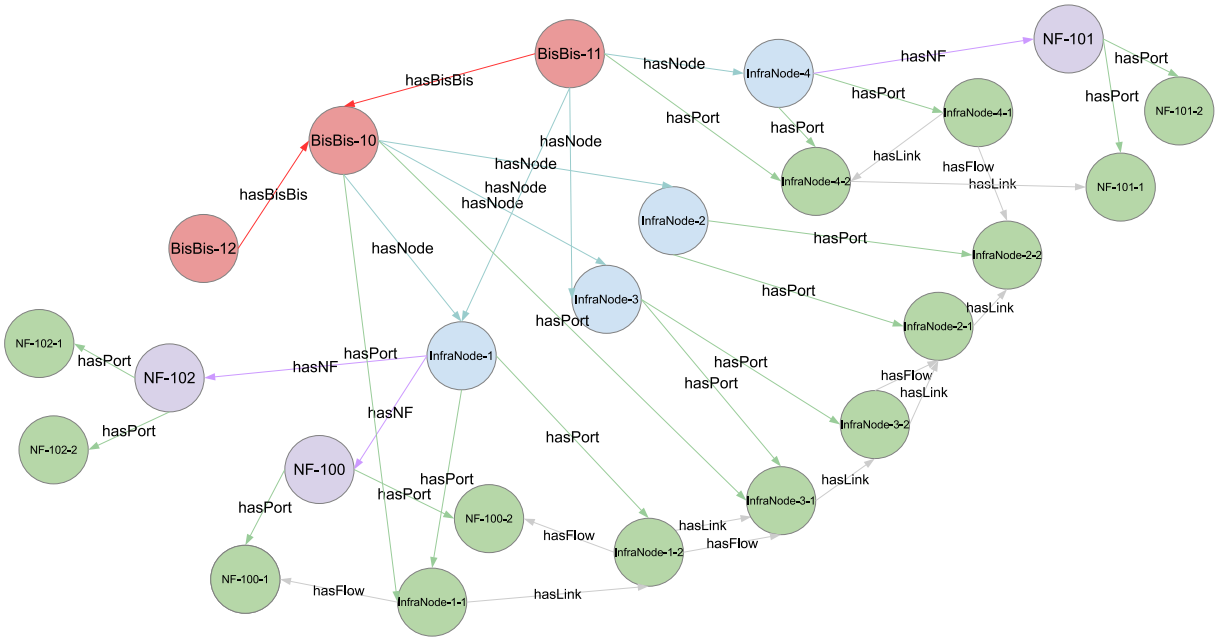

Figura 25 – Visão dos dados indexados no Neo4j

4.2.4 Análise dos Resultados

Assim como o capítulo anterior, não foram utilizadas métricas de desempenho na avaliação dos resultados. O cenário de virtualização recursiva do UNIFY foi reproduzido no Neo4j, bem como primitivas do consumo do orquestrador. Os dados de BiS-BiS, nós de infraestrutura e funções de rede foram armazenados e consultas de leitura foram implementadas.

Considerando o cenário naturalmente representado em grafo, do ponto de vista de complexidade, a modelagem foi simples. Para a escrita das consultas, a flexibilidade da linguagem Cypher permitiu buscar os relacionamentos conforme a necessidade das primitivas.

A primitiva de alocação por *tag* exigiu mais observação e testes, pois a restrição de consulta foi baseada nas propriedades dos relacionamentos *hasFlow*: *match* e *action*. Apesar disso, a primitiva foi parcialmente solucionada, o resultado não está apresentando as alocações separadamente, como um fluxo. Contudo, a utilização do Neo4j para essa aplicação se mostrou compatível e promissora no geral, e ainda se vislumbra para os trabalhos futuros, o refinamento dos resultados das alocações, e a exploração de novos cenários dentro do UNIFY, por exemplo, recursão de *Virtualizer*, composição de estruturas do *Virtualizer* e orquestração.

5 Conclusão e Trabalhos Futuros

O Capítulo 2 deste trabalho apresentou uma fundamentação teórica e os trabalhos relacionados das áreas de modelos semânticos, armazenamento em GDBs, arquitetura SDN e NFV que serviram de base e motivação para o desenvolvimento das propostas.

A principal contribuição da pesquisa é a proposta de utilização de GDBs, em particular com o modelo de grafo de propriedades, para a tarefa de gerenciamento de topologias, ou seja, no armazenamento e consulta de topologias de redes. Como cenários de caso de uso, para validar a proposta, foram considerados os controladores SDN e orquestradores NFV:

- **Mapeamento Semântico e Arquitetura**: conforme apresentado no Capítulo 3, a ontologia NML foi utilizada para modelar a topologia de uma rede SDN, e tais dados foram indexados no Neo4j. Além disso, as primitivas do NetGraph (RAGHAVENDRA *et al.*, 2012) foram reproduzidas em linguagem de consulta nativa Cypher. Foi apresentada a arquitetura dessa proposta de inserção, consulta e atualização dos dados, e também a formalização do *workflow* de *parsing* dos dados do modelo para o banco e vice-versa. A partir dos experimentos realizados, o NML se mostrou compatível com as necessidades de modelagem no cenário, entretanto uma limitação foi identificada, que o modelo não prevê características de custo para um *Link*. Para tal limitação foi apresentado um estudo inicial de extensão da ontologia NML. Essa extensão focou na modelagem de tabelas de roteamento e foram propostas classes, *object properties* e *data properties*. Entretanto, por se tratar de um estudo inicial a proposta não foi devidamente validada, deixando para os trabalhos futuros. Além disso, os resultados da avaliação experimental se mostraram promissores para os diferentes tamanhos de rede (grafo) e comparado com o RDBM possui vantagens de implementação.

- **Cenários de Avaliação**: dois cenários foram validados, Multidomínios SDN e Virtualização Recursiva. Em Multidomínios SDN, o cenário foi de troca de informações de topologia entre controladores de diferentes domínios descritos pelo ONF (ONF, 2015). A troca de informações de topologia é dada a partir de políticas previamente estabelecidades. O Neo4j foi utilizado para armazenar as visões exportadas dos diferentes controladores. Além disso, um grupo de primitivas do *Transport* API (ONF, 2015) foram reproduzidas em linguagem nativa Cypher. Em Virtualização Recursiva, o cenário do projeto Unify (UNIFY, 2016) foi validado. Os dados do BiS-BiS, responsável por agregar os nós de infraestrutura e funções de rede, foram inseridos no Neo4j. Consultas do estado do banco, alocações e geração de BiS-BiS foram escritas em linguagem nativa Cypher.

Para todos os casos de uso com o Neo4j, o seu modelo de grafo de propriedades

foi compatível com cada cenário. Os dados foram armazenados segundo a modelagem (entidades, atributos e relacionamentos) do NML e demais modelos, sem a necessidade de adaptações estruturais. Por fim, a linguagem de consulta Cypher se mostrou flexível e de fácil implementação, uma vez que permite busca de padrões conforme modelada. A utilização de GDB para o contexto de redes de computadores, se apresentou uma abordagem viável.

Pretende-se a continuação do trabalho nas seguintes direções:

- avaliar o desempenho com cargas de trabalho dinâmicas e aplicações no controlador OpenDaylight usando REST APIs;
- projetar otimizações na latência e capacidade do sistema via pré-calculo, uso de propriedades para habilitar ou desabilitar nós no grafo e compactação do mapeamento;
- validar a proposta de extensão do NML com as tabelas de roteamento, apresentada nesta dissertação;
- estudar novos casos de uso do UNIFY, tanto no banco de dados baseado em grafos quanto a aplicação do modelo semântico;
- desenvolver extensões do modelo semântico (NML/INDL) para refletir a descrição de redes SDN e suportar a modelagem de funções de rede virtualizadas (NFV) oferecidas como serviço;
- explorar as extensões do modelo semântico com raciocinadores, com o objetivo de realizar inferências e verificar inconsistências;

Referências

BERDE, P.; GEROLA, M.; HART, J.; HIGUCHI, Y.; KOBAYASHI, M.; KOIDE, T.; LANTZ, B.; O'CONNOR, B.; RADOSLAVOV, P.; SNOW, W.; PARULKAR, G. Onos: Towards an open, distributed sdn os. In: *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. New York, NY, USA: ACM, 2014. (HotSDN '14), p. 1–6. ISBN 978-1-4503-2989-7. Disponível em: <http://doi.acm.org/10.1145/2620728.2620744>. Citado 2 vezes nas páginas 34 and 35.

BOUZID, C. C. S.; PINATON, J. A survey of semantic web standards to representing knowledge in problem solving situations. p. 121–125, 2012. Citado 2 vezes nas páginas 19 and 20.

BRICKLEY, D.; GUHA, R. B. *RDF Schema 1.1*. 2014. Disponível em: <https://www.w3.org/TR/rdf-schema/>. Acesso em: 2 mar. 2016. Citado na página 20.

CARROZZO, G.; SZABO, R.; PENTIKOUSIS, K. *Network Function Virtualization: Resource Orchestration Challenges*. [S.l.], 2015. Disponível em: <https://tools.ietf.org/html/draft-caszpe-nfvrg-orchestration-challenges-00>. Citado na página 63.

CORPORATION, O. *MySQL*. 2015. Disponível em: <http://www.mysql.com/>. Acesso em: 23 set. 2015. Citado na página 53.

DIJKSTRA, F.; HAM, J. van der. *A URN Namespace for Network Resources*. 2013. GFD 202 (Community Practise). Disponível em: <http://www.ogf.org/documents/GFD.202.pdf>. Acesso em: 23 jan. 2016. Citado na página 41.

ESCALONA, E.; PENG, S.; NEJABATI, R.; SIMEONIDOU, D.; GARCIA-ESPIN, J. A.; FERRER, J.; FIGUEROLA, S.; LANDI, G.; CIULLI, N.; JIMENEZ, J.; BELTER, B.; DEMECHENKO, Y.; LAAT, C. de; CHEN, X.; YUKAN, A.; SOUDAN, S.; VICAT-BLANC, P.; BUYSSE, J.; LEENHEER, M. D.; DEVELDER, C.; TZANAKAKI, A.; ROBINSON, P.; BROGLE, M.; BOHNERT, T. M. GEYSERS: A novel architecture for virtualization and co-provisioning of dynamic optical networks and IT services. In: *2011 Future Network & Mobile Summit, Warsaw, Poland, June 15-17, 2011*. [S.l.: s.n.], 2011. p. 1–8. Citado 3 vezes nas páginas 22, 25, and 33.

ETSI. *Network Functions Virtualisation (NFV) - Architectural Framework, ETSI GS NFV 002 v1.2.1*. 2014. Disponível em: <http://www.etsi.org/technologies-clusters/technologies/nfv>. Acesso em: 2 mar. 2016. Citado 3 vezes nas páginas , 31, and 32.

ETSI. *Network Functions Virtualisation (NFV) - Terminology for Main Concepts in NFV, ETSI GS NFV 003 v1.2.1*. 2014. Disponível em: <http://www.etsi.org/technologies-clusters/technologies/nfv>. Acesso em: 2 mar. 2016. Citado na página 31.

ETSI. *Europe Telecomunications Standards Institute, Industry Specification Groups (ISG) - NFV*. 2015. Disponível em: <http://www.etsi.org/technologies-clusters/technologies/nfv>. Acesso em: 2 mar. 2016. Citado na página 31.

FOUNDATION, A. S. *Apache Jena*. 2016. Disponível em: <https://jena.apache.org/>. Acesso em: 2 mar. 2016. Citado na página 44.

GANDON, F.; SCHREIBER, G. *RDF 1.1 XML Syntax*. 2014. Disponível em: <https://www.w3.org/TR/rdf-syntax-grammar/>. Acesso em: 22 fev. 2016. Citado na página 20.

GHIJSEN, M.; HAM, J. van der; GROSSO, P.; DUMITRU, C.; ZHU, H.; ZHAO, Z.; LAAT, C. de. A semantic-web approach for modeling computing infrastructures. In: *Journal of Computers and Electrical Engineering*. [S.l.: s.n.], 2013. v. 39, p. 2553–2565. Citado 3 vezes nas páginas , 22, and 32.

GOMES, P. M.; ERVEN, R. C. G. S. A. V. *Publicação de Dados Governamentais Abertos Utilizando Tecnologias da Web Semântica*. 2011. Citado 2 vezes nas páginas 19 and 20.

GROSSO, P.; HERR, L.; OHTA, N.; HEARTY, P.; LAAT, C. de. Cinegrid: Super high definition media over optical networks. *Future Gener. Comput. Syst.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 27, n. 7, p. 881–885, jul. 2011. ISSN 0167-739X. Disponível em: <http://dx.doi.org/10.1016/j.future.2011.03.003>. Citado 3 vezes nas páginas 22, 25, and 33.

HAM, J. van der; DIJKSTRA, F.; LAPACZ, R.; ZURAWSKI, J. *Network Markup Language Base Schema version 1*. [S.l.], 2013. Citado 5 vezes nas páginas , 39, 41, 47, and 48.

HAM, J. van der; DIJKSTRA, F.; LAPACZ, R.; BROWN, A. The network markup language (nml) a standardized network topology abstraction for inter-domain and cross-layer network applications. In: *The TERENA Networking Conference (TNC)*. [S.l.: s.n.], 2013. Citado 3 vezes nas páginas 17, 21, and 22.

HITZLER, P.; KROTZSCH, M.; PARSIA, B.; SCHNEIDER, P. F. P.; RUDOLPH, S. *OWL 2 Web Ontology Language Primer (Second Edition)*. 2012. Disponível em: <https://www.w3.org/TR/2012/REC-owl2-primer-20121211/>. Acesso em: 23 fev. 2016. Citado na página 20.

HOLZSCHUHER, F.; PEINL, R. Performance of graph query languages: Comparison of cypher, gremlin and native access in neo4j. In: *Proceedings of the Joint EDBT-ICDT 2013 Workshops*. New York, NY, USA: ACM, 2013. (EDBT '13), p. 195–204. ISBN 978-1-4503-1599-9. Disponível em: <http://doi.acm.org/10.1145/2457317.2457351>. Citado 2 vezes nas páginas 27 and 36.

JOUILI, S.; VANSTEENBERGHE, V. An empirical comparison of graph databases. In: *Social Computing (SocialCom), 2013 International Conference on*. [S.l.: s.n.], 2013. p. 708–715. Citado 5 vezes nas páginas 26, 27, 35, 38, and 52.

KOPONEN, T.; CASADO, M.; GUDE, N.; STRIBLING, J.; POUTIEVSKI, L.; ZHU, M.; RAMANATHAN, R.; IWATA, Y.; INOUE, H.; HAMA, T.; SHENKER, S. Onix: A distributed control platform for large-scale production networks. In: *OSDI*. [S.l.: s.n.], 2010. v. 10, p. 1–6. Citado 2 vezes nas páginas 34 and 35.

KREUTZ, D.; RAMOS, F. M. V.; VERISSIMO, P.; ROTHENBERG, C. E.; AZODOLMOLKY, S.; UHLIG, S. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, v. 103, n. 1, p. 14–76, Jan 2015. ISSN 0018-9219. Citado 4 vezes nas páginas , 16, 28, and 29.

LAUER, G.; IRWIN, R. E.; KAPPLER, C.; NISHIOKA, I. Distributed resource control using shadowed subgraphs. *CoNEXT'13*, 2013. Citado na página 35.

LITERATEPROGRAMS. *Dijkstra's Algorithm (Java)*. 2015. Disponível em: <http://en.literateprograms.org/Dijkstra's_algorithm_(Java)/>. Acesso em: 20 set. 2015. Citado na página 54.

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1355734.1355746>. Citado 2 vezes nas páginas 16 and 29.

MEDINA, A.; LAKHINA, A.; MATTA, I.; BYERS, J. Brite: An approach to universal topology generation. In: *Proceedings of MASCOTS 01*. Washington, DC, USA: IEEE Computer Society, 2001. p. 346. Citado na página 49.

MIJUMBI, R.; SERRAT, J.; GORRICHO, J.-L.; BOUTEN, N.; TURCK, F. D.; BOUTABA, R. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys and Tutorials*, 2015. Citado 5 vezes nas páginas 16, 30, 31, 32, and 63.

MILLER, J. J. Graph database applications and concepts with neo4j. In: *Proceedings of the Southern Association for Information Systems Conference*. Atlanta, USA: [s.n.], 2013. Citado 4 vezes nas páginas , 16, 25, and 26.

NEO4J. *Neo4j Cypher Refcard*. 2015. Disponível em: <http://neo4j.com/docs/stable/cypher-refcard/>. Acesso em: 18 jan. 2016. Citado na página 28.

NOSQL. *NOSQL Databases*. 2016. Disponível em: <http://nosql-database.org/>. Acesso em: 23 jan. 2016. Citado na página 25.

NOVI. *Networking innovations Over Virtualized Infrastructures*. 2016. Disponível em: <http://www.fp7-novi.eu/>. Acesso em: 23 jan. 2016. Citado 3 vezes nas páginas 22, 25, and 33.

OMNES, N.; BOUILLON, M.; FROMENTOUX, G.; GRAND, O. L. A programmable and virtualized network & it infrastructure for the internet of things. *International Conference on Intelligence in Next Generation Networks*, 2015. Citado na página 30.

ONF. *Functional Requirements for Transport API*. [S.l.], 2015. Disponível em: <https://github.com/OpenNetworkingFoundation/ONFOpenTransport/blob/develop/TransportAPI/TAPI_FRS.13.pdf>. Citado 6 vezes nas páginas , 59, 60, 61, 68, and 78.

ONF. *Open Networking Foundation*. 2016. Disponível em: <https://www.opennetworking.org/about/onf-overview>. Acesso em: 8 fev. 2016. Citado na página 29.

ONF-TR-502. *SDN Architecture*. [S.l.], 2014. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports>. Citado 4 vezes nas páginas , 56, 57, and 58.

ONF-TR-512. *Technical Recommendation: Core Information Model version 1.0*. [S.l.], 2015. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports>. Citado na página 58.

ONF-TR-513. *Technical Recommendation: Common Information Model version 1.0*. [S.l.], 2015. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports>. Citado na página 58.

PANTUZA, G.; SAMPAIO, F.; VIEIRA, L. F.; GUEDES, D.; VIEIRA, M. A. Network management through graphs in software defined networks. In: IEEE. *Network and Service Management (CNSM), 2014 10th International Conference on*. [S.l.], 2014. p. 400–405. Citado 2 vezes nas páginas 34 and 35.

PENTEADO, R. R. M.; SCHROEDER, R.; HOSS, D.; NANDE, J.; MAEDA, R. M.; COUTO, W. O.; HARA, C. S. Um estudo sobre bancos de dados em grafos nativos. *X Escola Regional de Banco de Dados (ERBD2014)*, 2014. Citado na página 26.

PITTARAS, C.; GHIJSEN, M.; WIBISONO, A.; GROSSO, P.; HAM, J. van der; LAAT, C. de. Semantic distributed resource discovery for multiple resource providers. *Eighth International Conference on Semantics, Knowledge and Grids*, 2012. Citado na página 33.

PULKKINEN, T.; SALLINEN, M.; SON, J.; PARK, J.-H.; LEE, Y.-H. Home network semantic modeling and reasoning 2014. a case study. In: *Information Fusion (FUSION), 2012 15th International Conference on*. [S.l.: s.n.], 2012. p. 338–345. Citado na página 37.

RAGHAVENDRA, R.; LOBO, J.; LEE, K.-W. Dynamic graph query primitives for sdn-based cloudnetwork management. In: *Proceedings of Hot Topics in Software Defined Networks*. [s.n.], 2012. (HotSDN '12), p. 97–102. ISBN 978-1-4503-1477-0. Disponível em: <http://doi.acm.org/10.1145/2342441.2342461>. Citado 7 vezes nas páginas 17, 34, 35, 37, 39, 68, and 76.

ROBINSON, I.; WEBBER, J.; EIFREM, E. *Graph Databases*. [S.l.]: O'Reilly Media, Inc., 2013. ISBN 1449356265, 9781449356262. Citado 6 vezes nas páginas 16, 17, 26, 27, 38, and 53.

ROSA, R.; SIQUEIRA, M.; BAREA, E.; MARCONDES, C.; ROTHENBERG., C. E. *Minicurso: Network Function Virtualization: Perspectivas, Realidades e Desafios*. 2014. Citado na página 16.

SHADBOLT, N.; HALL, W.; BERNERS-LEE, T. The semantic web revisided. In: *IEEE Intelligent Systems*. [S.l.: s.n.], 2006. v. 21, p. 96–101. Citado 5 vezes nas páginas , 17, 19, 20, and 21.

SKöLDSTRöM, P.; KIND, M.; JOHN, W.; GARAY, J.; MATIAS, J.; JOCHA, S.; SZABô, R.; TAVERNIER, W. *Deliverable D3.2a - Network Function Forwarding Graph specification*. [S.l.], 2015. Disponível em: <https://www.fp7-unify.eu/files/fp7-unify-eu-docs/Results/Deliverables/>. Citado na página 65.

SOUNDARARAJAN, V.; KAKARADDI, S. Applying graph databases to cloud management: An exploration. In: *Cloud Engineering (IC2E)*. [S.l.: s.n.], 2014. p. 544–549. Citado 3 vezes nas páginas 36, 50, and 51.

SOUZA, T. P. C.; SANTOS, M. A. S.; PAULA, L. B.; ROTHENBERG, C. E. Modelos semânticos em bancos de dados baseados em grafos para aplicações de controle de redes definidas por software. *XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC2015): XX Workshop de Gerência e Operação de Redes e Serviço (WGRS*, 2015. Citado na página 75.

SOUZA, T. P. C.; SANTOS, M. A. S.; PAULA, L. B.; ROTHENBERG, C. E. Towards semantic networks models via graph databases for sdn applications. In: *Europe Workshop on Software Defined Networks (EWDSN 2015)*. [S.l.: s.n.], 2015. p. 49–54. Citado na página 75.

SZABO, R.; QIANG, Z.; KIND, M. *Towards recursive virtualization and programming for network and cloud resources*. [S.l.], 2015. Disponível em: <https://tools.ietf.org/html/draft-caszpe-nfvrg-orchestration-challenges-00>. Citado 2 vezes nas páginas 63 and 65.

SZABO, R.; SONKOLY, B.; KIND, M. *Deliverable 2.2: Final Architecture*. [S.l.], 2014. Disponível em: <http://www.fp7-unify.eu/index.php/results.html>. Citado 4 vezes nas páginas , 63, 64, and 65.

UNIFY. *Unify - Unifying cloud and carrier networks*. 2016. Disponível em: <https://www.fp7-unify.eu/>. Citado 2 vezes nas páginas 63 and 68.

W3C. *Inferency in Semantic Web*. 2016. Disponível em: <https://www.w3.org//standards/semanticweb/inference>. Acesso em: 23 jan. 2016. Citado na página 20.

W3C. *World Wide Web Consortium*. 2016. Disponível em: <https://www.w3.org>. Acesso em: 23 jan. 2016. Citado na página 19.

ANEXO A – Publicações

A realização dessa pesquisa resultou em duas publicações:

1. Artigo “Modelos Semânticos em Bancos de Dados Baseados em Grafos para Aplicações de Controle de Redes Definidas por Software” apresentado e publicado nos anais do XX Workshop de Gerência e Operação de Redes e Serviços (WGRS) do XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC) realizado em Vitória-ES em Maio de 2015(SOUZA *et al.*, 2015a).

2. Artigo “*Towards Semantic Network Models via Graph Databases for SDN Application*” apresentado e publicado nos anais do *Fourth European Workshop on Software Defined Networks (EWSDN)* realizado em Bilbao, Espanha em Outubro de 2015 (SOUZA *et al.*, 2015b).

ANEXO B – Consultas em Cypher - Primitivas

Conforme apresentado no Capítulo 3, as primitivas do NetGraph (RAGHAVENDRA *et al.*, 2012) foram escritas na linguagem de consulta Cypher, nativa do Neo4j. Abaixo serão apresentadas as consultas geradas e utilizadas pela aplicação de teste.

Grau de entrada de um *Node*:

```
1. MATCH (n:Node)<-[:hasInboundPort]-(p:Port)<-[isSink]-(l:Link)
2. WHERE n.name={nameNode}
3. RETURN COUNT(l) AS CountOutDegree
```

Grau de saída de um *Node*:

```
1. MATCH (n:Node)-[:hasOutboundPort]->(p:Port)-[isSource]->(l:Link)
2. WHERE n.name={nameNode}
3. RETURN COUNT(l) AS CountOutDegree
```

Vizinhos de um *Node*:

```
1. MATCH (n:Node)-[]-(p:Port)-[]-(l:Link)-[]-(p1:Port)-[]-(n2:Node)
2. WHERE n.name={nameNode}
3. RETURN DISTINCT(n2) AS Neighbors
```

Verificação da existência de rota entre dois *Nodes*:

```
1. MATCH p=shortestPath((n:Node)-[*]-(m:Node))
2. WHERE n.name = {nameNode} AND m.name={nameNode1}
3. RETURN COUNT(p) >0 AS DoesRouteExist
```

Cálculo de menor caminho de um *Node* para todos os outros:

```
1. MATCH (n:Node),(m:Node), p=shortestPath((n)-[*]->(m))
2. WHERE n.name={nameNode}
3. RETURN p
```

Cálculo de menor caminho de todos os pares de *Node*:

```
1. MATCH p=shortestPath((n:Node)-[*]->(m:Node))
2. RETURN p
```

Cálculo de *k* menores caminhos entre dois *Nodes*:

```
1. MATCH (n:Node),(m:Node), p=allShortestPaths((n)-[*]-(m))
2. WHERE n.name={nameNode} AND m.name={nameNode1}
3. RETURN p LIMIT {valueK}
```

Cálculo de *Minimum Spanning Tree* a partir de uma *Node* de origem:

```
1. MATCH p=shortestPath((n:Node)-[*]->(m:Node))
2. WHERE n.name = {nameNode}
3. RETURN p AS MinimumSpanningTree
```

Exclusão de um *Node* (e suas *Ports*):

```
1. MATCH (n:Node)-[r1]-(p:Port)-[r2]-(l:Link)-[r3]-(p2:Port)
2. WHERE n.name={nameNode}
3. DELETE n,r1,p,r2,l,r3
```

Atribuição de custo a um *Link*:

```
1. MATCH (n:Node)-[:hasOutboundPort]-()-[r]-(l:Link)
2. WHERE n.name={nameNode} AND l.name={nameLink}
3. SET l.cost={valueCost}
```

Busca de custo de um *Link*:

```
1. MATCH (n:Node)-[:hasOutboundPort]-()-[r]-(l:Link)
2. WHERE n.name={nameNode} AND l.name={nameLink}
3. RETURN l.cost
```

Inclusão de um *Node* (e suas *Ports*):

```
1. CREATE (n1:Node{name: newNode})
2. CREATE (n2:Port{name: portInNewNode})
3. CREATE (n3:Port{name: portOutNewNode})
4. WITH n1, n2, n3
5. CREATE (n1)<-[r:hasInboundPort]-(n2)
6. CREATE (n1)-[r2:hasOutboundPort]->(n3)
7. RETURN r,r2
```

ANEXO C – Consultas em Cypher - Multidomínios SDN

Conforme apresentado no Capítulo 4, as primitivas do caso de uso ONF (ONF, 2015) foram escritas na linguagem de consulta Cypher, nativa do Neo4j. Abaixo serão apresentadas as consultas geradas:

Topologias contidas em um FD:

```
1. MATCH (n:FD)-[r:encompas]-(m)
2. WHERE n.name={nameFD}
3. RETURN m AS getTopologyEncompassedByFD
```

Detalhes (portas) de um FD:

```
1. MATCH (n:FD)-[r:hasPort]-(m)
2. WHERE n.name={nameFD}
3. RETURN m AS getNodeDetails
```

Detalhes (*links*) entre dois FDs:

```
1. MATCH (n:FD)-[:hasPort]-(o)-[r]-(p)-[:hasPort]-(m:FD)
2. WHERE n.name = {nameFD1} AND m.name = {nameFD2}
3. RETURN o AS DetailsLink1, p AS DetailsLink2
```

Detalhes (*service endpoints*) de um FDs:

```
1. MATCH (n:FD)-[:hasService]-(m)
2. WHERE n.name = {nameFD}
3. RETURN m AS getServiceEndPointDetails
```

ANEXO D – Consultas em Cypher - Virtualização Recursiva

Conforme apresentado no Capítulo 4, as primitivas do caso de uso do Projeto Unify foram escritas na linguagem de consulta Cypher, nativa do Neo4j. Abaixo serão apresentadas as consultas geradas:

Conjunto de BisBis e seus *InfraNodes*:

```
1. MATCH (n:BisBis)-[:hasNode]-(m:InfraNode)
2. RETURN n AS BisBis, m AS InfraNode
```

Conjunto de *InfraNodes* e suas NFs:

```
1. MATCH (n:InfraNode)-[:hasNF]-(m:NF)
2. RETURN n AS InfraNode, m AS NF
```

Links entre portas e seus respectivos *InfraNodes*:

```
1. MATCH (i:InfraNode)-[:hasPort]->(n:Port)-[:hasLink]-
(m)<-[:hasPort]-(j:InfraNode)
2. Return i AS InfraNode1, n AS Port1, m AS Port2, j AS InfraNode2
```