Interfacer: Um método para especificação de interfaces para Internet do futuro

João Eurípedes Pereira Júnior

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2019

João Eurípedes Pereira Júnior

Interfacer: Um método para especificação de interfaces para Internet do futuro

Tese de doutorado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Prof. Pedro Frosi Rosa Ph.D.
Coorientador: Prof. Flávio de Oliveira Silva Ph.D.

Uberlândia
2019

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

P436i 2019

Pereira Júnior, João Eurípedes, 1978-
Interfacer [recurso eletrônico] : um método para especificação de interfaces para internet do futuro / João Eurípedes Pereira Júnior. - 2019.

Orientador: Pedro Frosi Rosa.
Coorientador: Flávio de Oliveira Silva
Tese (Doutorado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.te.2021.5510
Inclui bibliografia.
Inclui ilustrações.

1. Computação. I. Rosa, Pedro Frosi, 1959-, (Orient.). II. Silva, Flávio de Oliveira, 1970-, (Coorient.). III. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. IV. Título.

CDU:681.3

Gloria Aparecida - CRB-6/2047
Bibliotecária

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

ATA DE DEFESA

Programa de Pós-Graduação em:	Ciência da Computação				
Defesa de:	Tese, 08/2019, PPGCO				
Data:	12 de agosto de 2019	Hora de início:	**11:00**	Hora de encerramento:	**13:38**
Matrícula do Discente:	11423CCP004				
Nome do Discente:	João Eurípedes Pereira Júnior				
Título do Trabalho:	Interfacer: Um Método para Especificação de Interfaces para Internet do Futuro				
Área de concentração:	Ciência da Computação				
Linha de pesquisa:	Sistemas de Computação				
Projeto de Pesquisa de vinculação:	-				

Reuniu-se na sala 1B132, Bloco 1B, Campus Santa Mônica, da Universidade Federal de Uberlândia, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Ciência da Computação, assim composta: Professores Doutores: Rodrigo Sanches Miani - FACOM/UFU, Lásaro Jonas Camargos- FACOM/UFU, José Gonçalves Pereira Filho - INF/UFES, Eduardo Coelho Cerqueira - ITC/UFPA, Flávio de Oliveira Silva (Coorientador) e Pedro Frosi Rosa - FACOM/UFU, orientador do candidato.

Ressalta-se que o Prof. Dr. José Gonçalves Pereira Filho participou da defesa por meio de videoconferência desde a cidade de Vitória - ES e o Prof. Dr. Eduardo Coelho Cerqueira da cidade de Belém-PA . Os outros membros da banca e o aluno participaram *in loco*.

Iniciando os trabalhos o presidente da mesa, Prof. Dr. Pedro Frosi Rosa, apresentou a Comissão Examinadora e o candidato, agradeceu a presença do público, e concedeu ao Discente a palavra para a exposição do seu trabalho. A duração da apresentação do Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir o candidato. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o candidato:

Aprovado

Esta defesa faz parte dos requisitos necessários à obtenção do título de Doutor.

O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Lásaro Jonas Camargos**, **Professor(a) do Magistério Superior**, em 16/08/2019, às 13:14, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Rodrigo Sanches Miani**, **Professor(a) do Magistério Superior**, em 16/08/2019, às 16:25, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Pedro Frosi Rosa**, **Professor(a) do Magistério Superior**, em 17/08/2019, às 12:17, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Flávio de Oliveira Silva**, **Professor(a) do Magistério Superior**, em 03/09/2019, às 09:52, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Eduardo Coelho Cerqueira**, **Usuário Externo**, em 06/09/2019, às 18:25, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **José Gonçalves Pereira Filho**, **Usuário Externo**, em 26/10/2019, às 13:12, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1458446** e o código CRC **B5879E2D**.

Referência: Processo nº 23117.070867/2019-39 SEI nº 1458446

Às minhas filhas Laura e Lia.

Agradecimentos

Agradeço a Deus, o criador de tudo, por me permitir co-criar nesta singela obra. Aos meus pais João e Maria José, por nutrirem o milagre da vida com amor, dedicação e esperança. À minha querida esposa Lorenna, e às minhas amadas filhas Laura e Lia, pelos muitos momentos compartilhados, dos quais tirei força e inspiração. Ao meu tio Ronaldo, cujas portas do armário cheio de revistas de eletrônica deram acesso a um mundo completamente novo.

Agradeço aos colegas de trabalho, Fernanda, Marta, Amanda, Edmundo, Alisson, Felix, Luzencourt, Santana, Jeziel, Muriel e Paulo pelo apoio incondicional. Aos colegas pós-graduandos, Acrísio, Caio, Giovani, Ítalo, Natal, Willian, Alex, Romerson e Edmo pelo companheirismo. Agradeço ao Steffen Smolka pelas explicações sobre o código fonte da linguagem Frenetic e ao Tiago Prince Sales pelas explicações sobre OntoUML e Menthor.

Agradeço ao Prof. e também colega de trabalho, Luís Fernando Faina, por ensinar o caminho das pedras na pós-graduação. Ao Prof. Lásaro Jonas Camargos pelos comentários sobre a estrutura e qualidade desse texto, espero ter conseguido capturar suas ideias. Ao Prof. Rodrigo Sanches Miani, pelo acompanhamento e comentários sobre o trabalho que muito o enriqueceram. Ao Prof. Eduardo Coelho Cerqueira, pelos comentários sobre gerenciamento de redes de computadores que muito contribuíram com esse estudo. Ao Prof. José Gonçalves Pereira Júnior, pela atenção e pelos preciosos comentários sobre ontologia, que influenciaram os rumos dessa pesquisa. Agradeço ao Prof. Flávio de Oliveira Silva, pela prontidão, disposição e alegria na discussão das ideias, na edição e revisão dos artigos.

Agradeço ao Prof. Pedro Frosi Rosa, por acreditar que esse trabalho fosse possível, pela confiança de haver um destino, ainda que muitas vezes incerto, pela orientação majestosa, pela amizade sincera, pelos conselhos, pelas conversas, pelos muitos cafés com ideias anotadas em guardanapos.

Agradeço, em especial, ao Erisvaldo, secretário da pós-graduação, pelo constante incentivo e sorriso gratuito. E à todas as pessoas que direta ou indiretamente colaboraram para que eu aqui chegasse. Agradeço ainda, pela saúde, pelo alimento, pela água, pelo

sol e por todas as coisas simples que às vezes tomamos como nossas por direito.

“πάντες ἄνθρωποι τοῦ εἰδέναι ὀρέγονται φύσει. σημεῖον δ’ ἡ τῶν αἰσθήσεων ἀγάπησις· καὶ γὰρ χωρὶς τῆς χρείας ἀγαπῶνται δι’ αὑτάς, καὶ μάλιστα τῶν ἄλλων ἡ διὰ τῶν ὀμμάτων.”

(Ἀριστοτέλης)

Resumo

Esta tese investiga como interfaces de gerenciamento de rede com uma semântica de comunicação enriquecida podem ser especificadas e implementadas. A motivação para essa investigação surgiu do ambiente de rede *Modelo de Título de Entidade* (MTE)/*Entity Title Architecture* (ETArch), cujo objetivo era investigar a aproximação semântica entre as camadas do modelo *International Organization for Standardization* (ISO) para melhorar o *Quality of Service* (QoS)/*Quality of Experience* (QoE) a partir da compreensão das necessidades das entidades comunicantes e da adaptação da rede para atendê-las. Partindo do ponto que a ontologia *Web Ontology Language* (OWL) não foi concebida para descrever requisitos complexos de comunicação, questiona-se quais tipos de ontologias podem auxiliar a melhorar o nível semântico das interfaces de gerenciamento de rede e quais ferramentas de engenharia de software podem ajudar a automatizar o desenvolvimento dessas interfaces. Para materializar tal intento, primeiro se estabelece um fio condutor que interliga as atividades de modelar e arquitetar redes relacionando as redes tradicionais com iniciativas de pesquisa de Internet do Futuro e com as recentes tendencias de softwarização das redes. Segundo, investiga-se como empregar a ontologia de fundamentação UFO, concebida para lidar com semântica do mundo real, a esse ambiente de rede de teste. Em terceiro, especifica-se uma ontologia de domínio com OntoUML para representar uma interface de gerenciamento com semântica enriquecida para o ambiente MTE/ETArch. Por último, essa ontologia é usada para demonstrar como a integração das infra estruturas de modelagem e desenvolvimento pode auxiliar para que a interface reflita de maneira ágil qualquer alteração na ontologia. A principal contribuição deste trabalho é propor a elevação da precisão semântica de interfaces de gerenciamento e da operação da rede visando aliviar a carga de avaliação semântica por parte dos gerentes da rede. Percebe-se, que o caminho para se alcançar interfaces de gerenciamento de rede com uma semântica mais apurada passa pelas ontologias. Não obstante, suspeita-se que será necessário o emprego de outras teorias semânticas formais para estabelecer um arcabouço semântico, sendo que muito ainda há por investigar nesse sentido.

Palavras-chave: Interface. Especificação. Arquitetura. Modelagem. Ontologia. UFO. OntoUML. SDN. NFV. ETArch.

Abstract

This thesis investigates how network management interfaces with enriched communication semantics can be specified and implemented. The motivation for this reseach came from the network testbed MTE/ETArch, whose objective was to investigate a semantic approximation between layers of the ISO model to improve the QoS/QoE based on comprehension of the needs of communicating entities and network adaptation to meet them. Starting from the point that the OWL ontology was not designed for strong semantics specifications, it reseaches which types of ontology can help to improve the semantic level of network management interfaces and which software engineering tools can help automate the development of these interfaces. In order to materialize this intention, first, it establishes a link between the activities of modeling and architecting networks relating traditional networks with Future Internet research initiatives and with recent trends of network softwarization. Second, it investigates how to employ the foundation ontology UFO, designed to deal with real-world semantics, in this testbed network. Third, we specify a domain ontology using OntoUML to represent a semantically-enriched management interface for the MTE/ETArch environment. Finally, this ontology is used to demonstrate how the integration of modeling and development infrastructures can help interfaces to readly reflect any changes in the ontology. The main contribution of this work is to propose the increase of the semantic accuracy of management interfaces to alleviate the load of semantic evaluation over the network managers. It is noticed that the way to achieve network management interfaces with a more refined semantics goes through ontologies. Nevertheless, it suspects that it will be necessary to use other formal semantic theories to establish a semantic framework, and much remains to be investigated in this regard.

Keywords: Interface. Specification. Architecture. Modeling. Ontology. UFO. OntoUML. SDN. NFV. ETArch.

Lista de ilustrações

Lista de siglas

AI *Artificial Intelligence*

API *Application Programming Interface*

A-CPI *Application-Control Plane Interface*

ALM *Application Layer Multicast*

ASN.1 *Abstract Syntax Notation One*

BSS *Bussiness Support System*

BS *Base Station*

CIM *Common Information Model*

CLI *Command Line Interface*

DTS *Domain Title Service*

DTSA DTS *Agent*

DTSCP DTS *Control Protocol*

DHCP *Dynamic Host Control Protocol*

EBI *East Bound Interface*

EMF *Eclipse Modeling Framework*

EMP *Eclipse Modeling Project*

EMS *Element Management System*

EPC *Evolved Packet Core*

ETArch *Entity Title Architecture*

ETCP *Entity Title Control Protocol*

ETSI *European Telecommunications Standards Institute*

HSS *Home Subscriber Server*

IDE *Integrated Development Environment*

IEEE *Institute of Electrical and Electronics Engineers*

IETF *Internet Engineering Task Force*

IP *Internetworking Protocol*

ITU *International Telecommunication Union*

ISO *International Organization for Standardization*

MANO *Management and Orchestration*

MDA *Model Driven Architecture*

MDE *Model Driven Engineering*

MDTSA *Master DTSA*

MIB *Management Information Base*

ML *Machine Learning*

MOF *Meta-Object Facility*

MOFM2T MOF *Model to Text Transformation Language*

MTE *Modelo de Título de Entidade*

MTL *Model Transformation Language*

MME *Mobility Management Entity*

MEHAR *Mondial Entities Horizontally Addressed by Requirements*

NBI *North Bound Interface*

NFV *Network Functions Virtualization*

NFVI *NFV Infrastructure*

NETCONF *Network Configuration Protocol*

NE *Network Element*

OCL *Object Constraint Language*

ODCM *Ontology-Driven Conceptual Modeling*

OMG *Object Management Group*

ONF *Open Networking Foundation*

OntoUML *Ontology* UML

OSI *Open Systems Interconnection*

OSS *Operation Support System*

OWL *Web Ontology Language*

PoA *Point of Attachment*

PNE *Physical Network Elements*

QoE *Quality of Experience*

QoS *Quality of Service*

RESTCONF *REST Configuration Protocol*

SBI *South Bound Interface*

SDN *Software Defined Networking*

SMI *Structure of Management Information*

SNMP *Simple Network Management Protocol*

TOSCA *Topology and Orchestration Specification for Cloud Applications*

UFO *Unified Foundational Ontologies*

UML *Unified Modeling Language*

WBI *West Bound Interface*

VM *Virtual Machine*

VNF *Virtualization Network Function*

VIM *Virtualised Infrastructure Manager*

XMI XML *Metadata Interchange*

YANG *Yet Another Next Generation*

Sumário

Capítulo 1

Introdução

A intenção desta tese é investigar como interfaces de gerenciamento de rede com uma semântica de comunicação enriquecida podem ser especificadas e implementadas. A motivação para essa investigação surgiu a partir do ambiente de rede MTE/ETArch desenvolvido como uma iniciativa de pesquisa de Internet do Futuro, cujo objetivo era investigar a aproximação semântica entre as camadas do modelo ISO, para melhorar o QoS/QoE a partir da compreensão das necessidades das entidades comunicantes e da adaptação da rede para atendê-las. Partindo do ponto que a ontologia OWL não foi concebida para descrever requisitos complexos de comunicação, questiona-se quais tipos de ontologias podem auxiliar a melhorar o nível semântico das interfaces de gerenciamento de rede e quais ferramentas de engenharia de software podem ajudar a automatizar o desenvolvimento dessas interfaces.

Para materializar tal intento, primeiro se estabelece um fio condutor que interliga as atividades de modelar e arquitetar redes relacionando as redes tradicionais com iniciativas de pesquisa de Internet do Futuro com as recentes tendencias de softwarização das redes, de forma a demonstrar que apesar ocorrer em um ambiente que não é recente, a pesquisa pode ser estendida com facilidade. Segundo, investiga-se como empregar a ontologia de fundamentação *Unified Foundational Ontologies* (UFO), concebida para lidar com semântica do mundo real, a esse ambiente de rede de teste. Em terceiro, especifica-se uma ontologia de domínio usando a linguagem OntoUML para representar uma interface de gerenciamento com semântica enriquecida para o ambiente MTE/ETArch. Por último, essa ontologia é usada para demonstrar como a integração das infra estruturas de modelagem e desenvolvimento pode auxiliar para que a interface reflita de maneira ágil qualquer alteração na ontologia.

A principal contribuição deste trabalho é propor a elevação da precisão semântica das interfaces de gerenciamento e da operação da rede visando aliviar a carga de avaliação semântica por parte dos gerentes da rede e a visando aproveitar a tendencia de incorporar tecnologias de *Artificial Intelligence* (AI) e *Machine Learning* (ML) na automação e na segurança das redes softwarizadas. Percebe-se, que o caminho para se alcançar interfaces

de gerenciamento de rede com uma semântica mais apurada passa pelas ontologias. Não obstante, suspeita-se que será necessário o emprego de outras teorias semânticas formais para estabelecer um arcabouço semântico, sendo que muito ainda há por investigar nesse sentido.

1.1 Motivação

A intenção desta tese é explorar como a aproximação semântica pode ser implementada. O MTE (PEREIRA et al., 2011a) foi desenhado com base em uma análise dos problemas arquiteturais (SHENKER, 2006) da Internet com o objetivo de promover a aproximação semântica entre as camadas superiores e inferiores. Essa aproximação semântica está diretamente ligada aos esforços para elevar a qualidade dos serviços de rede ,QoS e a qualidade de experiência dos usuários, QoE, por exemplo, (JÚNIOR et al., 2013) e (LEMA et al., 2014). O MTE faz referência às camadas do modelo OSI, que divide o problema da comunicação em partes menores e as organiza em camadas.

Nesse modelo as camadas superiores ficaram responsáveis pelos requisitos de comunicação das aplicações, as camadas intermediárias ficaram responsáveis pelos requisitos da rede e as camadas inferiores ficaram responsáveis pelos requisitos do meio físico. Com o tempo, as camadas das extremidades evoluíram, enquanto que as camadas intermediárias permaneceram quase que originais, tornando mais difícil o casamento dos requisitos das aplicações e dos meios físicos com os requisitos da rede. Embora o MTE tenha proposto a aproximação semântica através da adoção de um esquema de endereçamento horizontal, ele não fornece diretrizes para a especificação desses requisitos de comunicação. Em suma, o problema de expressar requisitos de aplicação, de meio físico e de rede uns em função dos outros persiste.

Além disso, este estudo se destina a obter meios de expressar requisitos de comunicação para entidades de rede, que segundo o MTE são componentes lógicos, a saber, aplicações, conteúdos, nuvens, sensores, hospedeiros e usuários, cujas necessidades devem ser compreendidas e suportadas por serviços de rede. Essa iniciativa usou OWL (PEREIRA et al., 2011a) a fim de descrever formalmente os conceitos de entidade, título, necessidade, camada e serviço. Tais conceitos, assim como no modelo OSI, foram organizados para compreender e suportar as necessidades dessas entidades. Essas necessidades, embora exemplificadas como atraso, largura de banda, jitter, endereçamento, garantia de entrega, gerenciamento, mobilidade, QoS e segurança, não chegaram a ser formalizadas em OWL.

Ocorre que OWL é uma linguagem baseada em lógica computacional capaz de representar conhecimento, verificar sua consistência e executar deduções sobre esse conhecimento. E a força de sua semântica é insuficiente para representar conhecimento de tal complexidade. Embora o termo ontologia possa assumir diversos significados, chama-se

atenção aqui para dois significados em particular. Primeiro no sentido de modelos com foco em representar com acurácia um domínio de interesse e, segundo, como artefatos de raciocínio usados em inteligencia artificial. A principal diferença entre os dois reside na eficiência computacional e expressividade. Modelos conceituais são muito expressivos e impróprios para o raciocínio automatizado e, no entanto, OWL foi desenhada para ser computacionalmente eficiente, permitindo o raciocínio, deduções e consultas, porém com menos expressividade (SALES; SALES, 2014).

Espera-se com este trabalho, estabelecer as bases para projetar um Interfacer. Inspirado no componente controlador da SDN, idealiza-se um outro componente, batizado interfaceador (*Interfacer*), cuja função é prover uma interface Operador-Rede programável e flexível. Assim como foi necessário definir um switch programável e um protocolo de programação antes de definir um controlador, é necessário investigar quais teorias matemáticas e técnicas de engenharia de software podem auxiliar na especificação e implementação de um Interfacer. e seus usuários, facilitando a definição e o atendimento dos requisitos de comunicação. Um dos passos em direção a essa aproximação semântica foi a introdução do uso de ontologia como ferramenta para especificar o MTE, a fim de produzir uma especificação compartilhada, que possa ser interpretada da mesma maneira por diferentes partes interessadas.

Ao revisar a literatura, percebe-se que modelos e arquiteturas sempre estiveram presentes no dia a dia das redes. Esses termos, refletem boas práticas, estabelecidas em atividades que podem ser enquadradas na "engenharia de redes". Ao incorporar essas práticas, uma pesquisa anterior realizada pelo nosso grupo apresentou um modelo e sua respectiva arquitetura (MTE/ETArch) de rede, que guardam estreita relação com outras iniciativas de Internet do Futuro como (PETERSON et al., 2006), (GAVRAS et al., 2007), (FISHER, 2007) (GALIS et al., 2011). Essas iniciativas investigam arquiteturas de rede alternativas que possam substituir as atuais e resolver problemas como a ausência escalabilidade, mobilidade e segurança. Além disso, o estudo de Internet do Futuro é precursor de tecnologias bem estabelecidas como SDN e NFV. De modo que as melhorias pleiteadas nesta tese possuem relação com tecnologias atuais e fundamentais na área de redes.

Embora a ideia possa ser aplicada a outras iniciativas de Internet do Futuro e mesmo à arquiteturas mais recentes como SDN e NFV, limita-se as investigações ao escopo da ambiente MTE/ETArch. Essa restrição de escopo se justifica devido à complexidade do problema. A ausência de discussões a respeito da padronização da comunicação de gerenciamento no sentido do gerente para a rede pode ser compreendida, primeiro, pela definição de gramáticas livres de contexto capazes de reconhecer as configurações de rede enviadas através de interfaces de linha de comando *Command Line Interface* (CLI), segundo, porque apenas recentemente o fenômeno de softwarização das começou a tornar essa ideia viável.

Talvez, a importância desta pesquisa não fique evidente à primeira vista, pois trata-se de um assunto pouco discutido, apesar de sua relevância. Trata-se da padronização da comunicação de gerenciamento no sentido do gerente para a rede, visto que a comunicação de gerenciamento no sentido da rede para o gerente já possui algum grau de padronização, vide *Structure of Management Information* (SMI)/*Management Information Base* (MIB)/*Simple Network Management Protocol* (SNMP), e mais recentemente *Yet Another Next Generation* (YANG)/*REST Configuration Protocol* (RESTCONF)/*Network Configuration Protocol* (NETCONF). Em outras palavras, como padronizar o envio de configurações para a rede? Essa padronização deve permitir, inclusive, a configuração de uma hierarquia de gerencia, ou ainda, uma hierarquia de dispositivos sob autoridade de determinada gerencia. A título de exemplificação, pode-se imaginar uma hierarquia de provedores de serviço de rede, onde os provedores locais estão sujeitos às regras dos provedores regionais, que por sua vez estão sujeitos às regras de operadores de rede. Havendo, em cada nível de gerenciamento, uma hierarquia de dispositivos.

Atualmente, essa comunicação de configuração é realizada através de interfaces definidas de acordo com as necessidades de cada fabricante, dispositivo e nível de gerenciamento. Posteriormente, essas configurações são traduzidas ou interpretadas para especificações mais formais e mais legíveis para máquinas, ditadas por organizações de padronização como o *Internet Engineering Task Force* (IETF) e o *Institute of Electrical and Electronics Engineers* (IEEE). Um projeto de uma infraestrutura de rede é guiado por dois fatores: requisitos e interfaces. Enquanto os requisitos são ditados pelos usuários e operadores da rede, menciona-se aqui três tipos de interfaces relevantes para um projeto.

Primeiro, a interface entre o usuário e a rede (Usuário-Rede), segundo, a interface entre o operador e a rede (Operador-Rede), e por último, a interface entre os dados e a rede (Dados-Rede) (CASADO et al., 2012). A interface Dados-Rede tem sido definida em termos de delimitação dos dados em quadros e cabeçalhos. O SDN adicionou uma abordagem programável à interface Dados-Rede nos nós, juntamente com uma visão lógica centralizada, tornando a rede mais flexível e ágil para manipular os dados. Some-se a isso, o esquema de endereçamento horizontal do MTE que reduziu a quantidade de delimitações simplificando essa interface. No entanto, nenhuma dessas iniciativas melhorou a questão da padronização da interface Operador-Rede.

Assim, a necessidade de elevar a expressividade na formalização do MTE, além da conveniência de que o conhecimento representado por tais modelos seja transferido para a ETArch de forma rápida e eficiente, se apresenta como motivação e como uma oportunidade de alinhar a pesquisa local com o que vem acontecendo em âmbito global, onde linguagens de modelagem como *Unified Modeling Language* (UML), YANG e *Topology and Orchestration Specification for Cloud Applications* (TOSCA) têm sido usadas para especificar configuração de dispositivos, interfaces para arquiteturas de rede e configuração de aplicações em nuvem.

Enquanto algumas interfaces das tecnologias SDN e NFV seguem padrões definidos por organizações de padronização, como a *Open Networking Foundation* (ONF) e o *European Telecommunications Standards Institute* (ETSI), nota-se a necessidade de processos de especificação e implementação de interfaces específicas (CASADO; FOSTER; GUHA, 2014) que se apoiem nas interfaces genéricas. Do mesmo modo, nota-se a necessidade de mecanismos capazes de gerenciar e elevar sistematicamente o nível das abstrações (TROIS et al., 2016), afim de acelerar o provisionamento de serviços sobre redes que utilizem essas tecnologias.

Logo, a capacidade de especificar, desenvolver e integrar interfaces específicas para serviços que devem trabalhar em conjunto com as interfaces definidas por organizações de padronização pode se tornar uma melhoria no processo de projeto e desenvolvimento de redes definidas por software. Opta-se, portanto, em seguir uma tendência que vem norteando diversas iniciativas como *Common Information Model* (CIM), YANG e TOSCA, que é o uso de técnicas de modelagem como ferramentas auxiliares no processo de especificação de arquiteturas e implementação de software.

O uso de modelagem para promover a padronização e impulsionar o desenvolvimento de diversos aspectos das arquiteturas SDN e NFV demonstram que essa técnica tem ganhado cada vez mais importância na área de redes de computadores. Destacam-se aqui a linguagem de configuração de dispositivos YANG, a linguagem de especificação de serviços em nuvem TOSCA e a iniciativa de modelagem CIM da ONF para modelar interfaces.

Técnicas modernas de modelagem, como a modelagem conceitual orientada a ontologia, além de melhorar a comunicação e o aprendizado dos envolvidos em projetos, implementações e implantações de sistemas, também permitem que conhecimentos sejam formalizados e utilizados por máquinas no processo de desenvolvimento. No entanto, resta investigar o comportamento dessas novas técnicas de modelagem em conjunto com novas tecnologias de rede, tais como SDN e NFV.

1.2 Objetivos

Este trabalho tem como objetivo investigar o uso de ontologias como forma de elevar o nível semântico na comunicação de gerenciamento de rede. Usa-se modelagem conceitual orientada a ontologia como ferramenta para especificar e implementar uma interface de gerenciamento de alto nível de abstração para um ambiente de rede de teste.

1.2.1 Objetivos Específicos

1. Estabelecer uma relação entre as atividades de modelar e arquitetar redes tradicionais com a modelagem de software no âmbito das redes softwarizadas;

2. Investigar o uso da ontologia UFO como suporte semântico para a comunicação de gerenciamento de rede e como suporte para o desenvolvimento de interfaces de gerenciamento de rede;

3. Especificar uma ontologia para prover semântica na comunicação de gerenciamento de rede, principalmente dos requisitos de comunicação das entidades do ambiente de rede MTE/ETArch, usando a linguagem de modelagem conceitual OntoUML;

4. Demonstrar a integração de um ambiente de modelagem com um ambiente de desenvolvimento a fim de alcançar o desenvolvimento ágil de interfaces de gerenciamento de rede com semântica forte;

1.3 Organização da Tese

Este documento está estruturado da seguinte forma:

- O Capítulo 2 apresenta o modelo MTE e a arquitetura de rede ETArch, mostrando quais são as relações com um grupo de pesquisas sobre rede conhecido como Internet do Futuro. Além disso, ele discorre sobre os tradicionais conceitos de modelo e arquitetura de rede, para estabelecer um elo entre a Internet, as pesquisas de Internet do Futuro e pesquisas sobre tecnologias mais recentes como SDN ou NFV.

- O Capítulo 3 apresenta uma investigação sobre a metodologia de modelagem conceitual orientada a ontologia *Ontology-Driven Conceptual Modeling* (ODCM). Essa metodologia é candidata a suporte semântico para a comunicação de gerenciamento de rede e candidata a suporte para o desenvolvimento de interfaces de gerenciamento de rede. Especificamente, utiliza-se a ontologia unificada de fundamentação UFO e a linguagem de modelagem OntoUML.

- O Capítulo 4 descreve o Interfacer, um método que utiliza a linguagem de modelagem conceitual OntoUML associada a processos de desenvolvimento orientado a modelos como o *Model Driven Engineering* (MDE) e o *Model Driven Architecture* (MDA) usando um ambiente EMF para especificar e implementar interfaces de arquiteturas de rede.

- O Capítulo 5 documenta a implementação de uma versão preliminar da interface de gerenciamento de rede com uma semântica enriquecida. Essa interface se destina a realizar os primeiros teste no ambiente de rede MTE/ETArch. Uma ontologia em linguagem OntoUML é usada para representar conhecimento transformado em código fonte usando a ferramenta Acceleo *Model Transformation Language* (MTL) do ambiente de desenvolvimento *Eclipse*.

- O Capítulo 6, por fim, traz as considerações finais do trabalho, enumerando as principais contribuições, dificuldades e as propostas de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta os fundamentos conceituais sobre os quais se assenta a tese ora apresentada, bem com, faz uma análise de trabalhos correlatos que servem para demonstrar que o assunto em tela é pertinente para o momento da computação distribuída. Além de apresentar, este capítulo evidencia como conceitos e trabalhos correlatos se relacionam com a presença tese.

A seção 2.2 introduz aspectos conceituais e filosóficos do Modelo de Título de Entidade (doravante 'Modelo de Título' ou simplesmente MTE) para as redes que têm sido genericamente denominadas como a próxima geração de internet. A seção 2.3 introduz os aspectos arquiteturais relativos a uma materialização do Modelo de Título.

2.1 Modelos e Arquiteturas de Redes de Computadores

A palavra arquitetura é usada para designar o resultado da ação de um arquiteto (construtor chefe), alguém que dispõe componentes de forma proposital, que usa seu conhecimento para causar boa impressão ou gerar funcionalidade. Arquiteturas são geralmente instanciadas a partir de modelos ou fragmentos bem estabelecidos dessa experiência ou conhecimento.

Embora o modelo de camadas tenha sido criado bem antes, um bom exemplo é o Modelo de Referência OSI (*Open Systems Interconnection Reference Model*), que serviu de inspiração a diversas arquiteturas, inclusive a arquitetura Internet. A análise percorre arquiteturas de Internet do Futuro, quem tentam resolver problemas relevantes da Internet atual. Esse esforço traz novas percepções que culminam no surgimento de SDN. Nesse ínterim, são publicados o Modelo de Títulos e a arquitetura ETArch.

2.1.1 O modelo de referência OSI

Quando as redes alcançaram abrangência internacional, a preocupação com padronização se tornou uma questão importante. Surgiram, então, os primeiros esforços para transformar essa preocupação em algo mais palpável. Em 1970, o CCITT (agora chamado de *International Telecommunication Union* (ITU)) e o ISO já organizavam os primeiros grupos de trabalho com a tarefa de produzir recomendações de padronização para interconexão de redes de dados em âmbito internacional. O modelo de referência OSI provê uma base comum para coordenar o desenvolvimento de padrões para a interconexão de sistemas, permitindo ainda que padrões existentes sejam colocados em perspectiva, com um modelo completo de referência. Também identifica áreas para desenvolvimentos e melhorias de padrões, e provê uma referência comum para manter a consistência entre todos os padrões relacionados.

A ideia básica é que cada camada adicione valor a serviços fornecidos pelo conjunto de camadas inferiores, de forma que todos os serviços necessários para rodar uma aplicação distribuída estejam disponíveis. Camadas também ajudam a simplificar o problema, pois o dividem em pedaços menores, e garantem a independência entre camadas ao definir quais serviços cada uma deve prover, além de garantir flexibilidade sobre como os serviços serão implementados ao definir como eles serão acessados. A justificativa do modelo OSI para o uso de sete camadas influenciou muitas arquiteturas posteriores, embora a quantidade de camadas esteja diminuindo gradativamente.

2.1.2 Arquitetura Internet

A arquitetura Internet pode ser representada pela organização em camadas do modelo OSI, com algumas simplificações. À primeira vista, duas camadas do modelo OSI são suprimidas, aquelas relativas às camadas de Sessão e de Apresentação. Além de especificar menos camadas, a arquitetura também reduziu o número de protocolos em suas camadas inferiores, tendo como filosofia a simplicidade de operação (BUSH; MEYER, 2002). Chama a atenção o fato de o protocolo IP ser frequentemente visto como a cintura fina da arquitetura Internet, isto é, novos protocolos foram adicionados às camadas Física e de Aplicação, a camada de Rede permanece praticamente intacta nos últimos 40 anos. Os protocolos IP (POSTEL, 1980a) e TCP (POSTEL, 1980b) prestam os serviços mais importantes da pilha, tanto que em muitas literaturas são referidos como arquitetura TCP/IP.

Existe uma estreita relação entre a técnica de modelagem por conceitualização e a forma de especificação dos padrões da Internet (RFC – *Request For Comments*). Essas Solicitações (*Requests*) permitem que novas ideias (ou soluções) sejam apresentadas à comunidade por meio de RFCs. Os detalhes para o tratamento desses documentos são descritos no RFC (POSTEL, 1982). Esses documentos permitem que novos conceitos

e tecnologias sejam apresentados à comunidade, os quais são testados através de experimentos, e os resultados desses experimentos devolvidos na forma de comentários, que alteram ou validam as proposições. Até o presente momento já foram publicados mais de 8.500 RFCs.

2.1.3 Internet do Futuro

A Internet fez um surpreendente e dramático sucesso. Projetada originalmente para permitir o compartilhamento de um pequeno grupo de pesquisadores, a Internet passou a ser usada por muitos milhões de pessoas. No entanto, aplicações multimídia, com novas características de tráfego e requisitos de serviço, impuseram um desafio interessante aos fundamentos técnicos da Internet (SHENKER, 2006). O projeto original dos protocolos TCP/IP especificava um ambiente no qual usuários finais eram mutuamente confiáveis, enquanto assumia que a rede era inerentemente não confiável (sem garantia de entrega), devido à possibilidade dela ser atacada fisicamente, destruindo seus roteadores e enlaces. Desde então, o ambiente mudou significativamente como efeito colateral de seu grande sucesso.

O aumento de usuários e o surgimento de novas aplicações levaram ao surgimento de problemas ou limitações, tais como: transmissão Multicast, Mobilidade, Multi-homing, QoS, Segurança, Perda de conectividade fim-a-fim, Ausência de autenticação, Privacidade e Contabilidade, Tráfego não solicitado, Eficiência energética, entre outros. Muitas arquiteturas de Internet do Futuro (AHLGREN et al., 2006) se debruçaram sobre os problemas enfrentados, que levaram a uma série de propostas de arquiteturas de redes que substituiriam a arquitetura Internet (NIKANDER; GURTOV; HENDERSON, 2010).

A arquitetura Internet apresenta diversos problemas e limitações, sendo que (NIKANDER; GURTOV; HENDERSON, 2010) elenca os cinco maiores problemas. É curiosa a argumentação dos autores, que chegam à conclusão que o tráfego não solicitado é um dos maiores problemas da Internet, pelos seguintes fatores:

- Uma abordagem arquitetural onde cada destinatário tem um nome explícito e cada remetente pode enviar pacotes sem o consentimento do destinatário;
- Uma estrutura de negócio onde o custo marginal do envio de pacotes adicionais tende a zero;
- A ausência de leis, tratados internacionais e, especialmente, estruturas que garantam a punição efetiva de quem se envolva em atividades ilegais na Internet;
- A ganância, própria da natureza humana, que leva algumas pessoas a comportamentos anti-éticos na esperança de obter lucro fácil.

Não há o que fazer em relação ao último aspecto, que, portanto, está fora do escopo desta análise. O terceiro aspecto é claramente regulatório e, apesar de não fazer parte do escopo deste trabalho, traz um requisito de que a rede seja capaz de responder a aspectos regulatórios tais como auditorias, registros (*logging*) e rastreamentos.

Os dois primeiros aspectos podem ser usados para criar arquiteturas que exijam o consentimento de destinatários, sem o qual os remetentes não podem enviar mensagens ou, no máximo, podem enviar mensagens de sinalização. A perda da relação fim-a-fim, que se refere ao problema criado ao usar tradução de endereços de rede (NAT – *Network Address Translation*), para resolver o problema de exiguidade de endereços IP, agravou o problema da superpopulação. NAT possibilita a reutilização de determinados endereços, para endereçar hospedeiros internos, ilimitadamente. Com isto, a identificação de hospedeiros originários de mensagens se tornou uma tarefa complexa. Por outro lado, a presença de agentes de mapeamento esconde o interlocutor e torna a rede uma caixa preta.

A dificuldade para implementar mobilidade e multi-conexão nos remete a outro problema de endereçamento. O acúmulo das tarefas de identificação e localização suscitam muitas propostas para separá-las. Além disso, não se trata mais, apenas, de identificar e localizar dispositivos, mas sim de identificar e localizar pessoas, uma vez que as pessoas estão cada vez mais dependentes dos dispositivos.

Endereçamento é um aspecto arquitetural crítico da Internet, mas alterá-lo implicaria em perder a compatibilidade com todo o parque instalado. O primeiro problema de endereçamento na Internet foi a escassez de endereços. Uma das soluções encontradas foi uma nova versão do protocolo IP, a IPv6 (WU et al., 2013), (DEERING, 1998), outra solução foi a tradução de endereços (SRISURESH; EGEVANG, 2000). No entanto, logo percebeu-se que há outros problemas que inviabilizam o esforço para implantar essas propostas.

Quando a Internet foi concebida era necessário identificar apenas os hospedeiros, além de não haver necessidade de mobilidade. Com o objetivo de preservar a simplicidade, adotou-se então um modelo de endereçamento com organização hierárquica e endereços com sobrecarga semântica, isto é, um endereço desempenha duas funções, a de identificador e a de localizador. Ironicamente, a simplicidade que facilitou a adoção da rede e permitiu um crescimento rápido, trouxe também novas demandas, que mostraram que a separação dessas funções é um dos caminhos para implementar novos requisitos.

Várias RFCs propõem a separação das funções de identificação e localização, como o *Locator/ID Separation Protocol* (LISP) (FARINACCI et al., 2013), *Identifier-locator Network Protocol* (ATKINSON; BHATTI, 2012). Em (QUOITIN et al., 2007), os autores avaliam os benefícios da separação proposta por LISP e, é possível perceber, o quanto é difícil promover alterações na cintura fina da pilha TCP/IP e manter a compatibilidade com o legado.

2.2 Modelo de Título de Entidade

O MTE aborda questões de comunicações de sistemas distribuídos em um alto nível de abstração e se propõe a modelar tais questões para solucionar problemas da arquitetura Internet. A premissa é inspirada nas dificuldades que a Internet enfrenta para entregar mobilidade, difusão seletiva (*Multicasting*), qualidade de serviço (QoS) nas comunicações, entre outras, como motivação para criar o modelo. Argumenta-se que tais problemas se devem em parte à ausência de mecanismos que: i) permitam a compreensão das necessidades de comunicação; ii) estabeleçam mecanismos de comunicação que atendam essas necessidades; e, ainda, iii) monitorem essas necessidades e sejam capazes de reagir de maneira a manter as comunicações e suas respectivas necessidades. Por outro lado, problemas arquiteturais da Internet, como o engessamento das camadas intermediárias da pilha TCP/IP e a adição de grande quantidade de novos protocolos às extremidades da pilha, como forma de contornar esse engessamento, são também parte integrante do problema.

O Modelo de Título propõe, então, a utilização de ontologia para representar formalmente essas necessidades de comunicação. Essa representação formal pode ser usada como base para a definição de mecanismos de compreensão, estabelecimento de canais de comunicação, monitoramento e reação da rede, guiados pelas necessidades de comunicação. Além disso, o modelo propõe a adoção de um esquema de endereçamento horizontal, de maneira a dar mais importância ao usuário, visto que, apesar de o usuário ser o real motivador para a existência de comunicações, o desenho da Internet se baseia fortemente em hospedeiros (*hosts*). O endereçamento horizontal reduz a complexidade do esquema de endereços e facilita a mobilidade, dado que não depende da localização física do portador do endereço. A arquitetura Internet especifica vários tipos de endereços para comunicações, sendo que cabe mencionar que pesa contra o endereço de sua camada de rede a sobrecarga de identificador e localizador de hospedeiros. Ao atrelar a localização ao endereço, o que caracteriza o endereçamento hierárquico, a arquitetura Internet inviabiliza uma das propriedades fundamentais das aplicações futuras, que é a mobilidade.

Em adição ao binômio composto pela 'representação ontológica dos requisitos de comunicação' e pelo 'esquema de endereçamento horizontal', o MTE é estruturado em camadas, aplicando-se regras de modelagem semelhante àquelas utilizadas na concepção do Modelo de Referência OSI. Essas regras visam garantir, ao mesmo tempo, a consistência do todo e a evolução das partes, mantendo baixo acoplamento entre as camadas do modelo e garantindo assim o isolamento de funções e a definição de suas interfaces. A estruturação em camadas permite uma comparação direta entre as pilhas de camadas da arquitetura Internet e do Modelo de Título de Entidade. A camada no topo da pilha reúne todas as funções relacionadas a Entidades comunicantes. O MTE define o conceito de Título para identificar Entidades, que apresentam similaridades àquelas da camada de Aplicação no modelo OSI. O MTE utiliza a identificação por meio de Título para tudo aquilo que tem

capacidade de se comunicar e precisa ser identificável em um sistema distribuído – que é a razão de ser do nome do modelo.

É importante lembrar, como já mencionado, que usuários são o motivo pela qual as comunicações existem, sendo que seus requisitos (de usuários) são atendidos apenas em parte por restrições arquiteturais da Internet. Requisitos tais como segurança e mobilidade são não atendidos, ou são atendidos em parte, por conta da sobrecarga semântica do endereço *Internetworking Protocol* (IP). Por exemplo, ao utilizar o IP como identificador e endereço, o requisito de mobilidade fica seriamente comprometido.

Assim, o Modelo de Título oferece serviços com suporte semântico a necessidades de comunicação, que, sobreposto a uma infra-estrutura de comunicação, é uma estratégia para mitigar dificuldades que as redes atuais enfrentam para prover qualidade de serviço e qualidade de experiência a suas comunicações. O Modelo de Título (PEREIRA; KOFUJI; ROSA, 2009) e (PEREIRA et al., 2011b) especifica os aspectos filosóficos de comunicações a partir da definição de três conceitos essenciais:

2.2.1 Título

Pode-se dizer que um dos maiores entraves à evolução da arquitetura Internet é a sobrecarga semântica que o endereço IP assumiu em termos de Identificação e Localização. As reflexões que levaram ao Modelo de **Título** de Entidade demonstraram que a identificação de aspectos da comunicação deve ser inequívoca, unívoca e independente da localização e da natureza da infraestrutura de comunicação.

O conceito de **Título** foi definido no Modelo de **Título** de Entidade para identificar elementos de comunicações que precisam ser identificáveis durante seus ciclos de vida no ambiente distribuído. **Títulos** são, portanto, identificadores abstratos, unívocos, não ambíguos e independentes da localização do elemento que identificam. Ao ser abstratos, e portanto de um alto nível de representação, **Títulos** independem da topologia e da natureza da infraestrutura de rede que suportam os elementos identificáveis.

A independência de localização habilita o Modelo de **Título** de Entidade a suportar uma das propriedades mais queridas em comunicações, que é a Mobilidade. A se mover ao longo da infraestrutura de comunicação, o **Título** permanece imutável e, portanto, suas identificação, autenticação e necessidades requeridas são facilmente verificadas.

2.2.2 Entidade

O Modelo de Título de **Entidade** enriquece o conceito de entidade definida no modelo de referência OSI, que define entidade todo **ser** que tem capacidade de se **comunicar** num ambiente distribuído. De fato, a semântica de Entidade se origina no Grego, para se referir a um 'ser que existe'. No MTE, **Entidades** têm necessidades para suportar suas comunicações, podem herdar propriedades e manter relações com outras entidades.

Como seres comunicantes, no Modelo de Título, **Entidades**, além de necessidades, têm capacidades que podem ser utilizadas em contextos de comunicação.

Entidades podem se relacionar com outras entidades em relações que podem ser: **Unicast** – envolvendo um par de entidades, numa relação uma para uma; **Multicast** – envolvendo um conjunto de entidades, numa relação uma para muitas; ou **Broadcast** – numa relação uma para todas. Olhando-se para estas definições, alguém menos atento, poderia pensar nos tipos de endereçamento que existem nas redes atuais. Todavia, é interessante lembrar que estes tipos de endereços foram desenvolvidos para suportar a necessidades de comunicações de usuários.

O conceito de **Entidade** definido no Modelo de Título permite a satisfação de diferentes conjuntos de requisitos, propostos por diferentes orientações de usos, tais como, redes orientadas a Serviços (*Service Centric*), Usuários (*User Centric*), Conteúdos (*Content Centric*), Dispositivos (*Device Centric*) etc.

No Modelo de Título, uma *Entidade* é univocamente identificada por um ou mais Títulos. Deste modo, um Título identifica unicamente uma **Entidade**, que pode ser identificada por diversos Títulos. Esta propriedade da identificação permite que **Entidades** possuam Títulos representativos no contexto no qual se encontram. **Entidades** são identificáveis e podem ser referenciadas por quaisquer de seus Títulos, independentemente de sua localização e da natureza da infra estrutura de comunicação.

2.2.3 *Workspace*

Embora, para se materializar, a comunicação exija o uso de aplicações instaladas em dispositivos, *facto*, o sujeito são entidades, por exemplo, a comunicação pode se dar de pessoa(s) para pessoa(s). Quando uma pessoa deseja se comunicar, há uma metonímia, ou seja, ela não pensa no celular que a outra pessoa carrega, nem na aplicação que roda dentro dele, a impressão que se tem é que a conversar se dá diretamente.

Evidentemente, nos dias atuais, há muitas comunicações que não envolvem pessoas, mas, ainda assim, desenvolvedores abstraem aspectos de comunicações. A arquitetura Internet é ubíqua em todos os campos de atividades e segmentos industriais. Mesmo em chão de fábrica, que muitas vezes envolve tempo real, há comunicações baseadas em TCP/IP.

Entidades precisam de domínios específicos de comunicação que suportem as necessidades requeridas e respeitem as capacidades das entidades envolvidas. Por exemplo, imagine um domínio de comunicação que envolva a transmissão de um video, por exemplo um filme. Veja que em um contexto, pode ser necessária uma capacidade 4k, enquanto que em outro momento, este mesmo video poderia ser visto a partir de um telefone celular.

O Modelo de Título de Entidade define o conceito de ***Workspace*** para representar o domínio de comunicação capaz de suportar as necessidades de comunicação de Entidades

participantes. ***Workspaces*** são capazes de suportar as relações de entidades (*Unicast*, *Multicast*, *Broadcast*) de forma natural.

Nas comunicações das redes atuais, embora exista a implementação de endereços *Multicast* e *Broadcast*, nas camadas de Enlace Lógico (*Logical Link Layer*) e de Rede (*Network Layer*), eles são limitados e complexos de serem usados, o que justifica que todas as aplicações que tenham esta natureza de relacionamento façam uso de retransmissões para realizar suas comunicações. Para minimizar, foram criadas estratégias de difusão na camada de aplicação (*Application Layer Multicast* (ALM)), que, contudo, não eliminaram as retransmissões.

Na realidade, o conceito de ***Workspace*** define que o modo de relacionamento normal é o *Multicasting*, sendo os demais, casos particulares. Quando se diz que o ***Workspace*** permite esses relacionamentos entre entidades de forma natural, significa que não há retransmissões, isto é, uma entidade envia uma mensagem e, a mensagem enviada, é entregue a todas as entidades participantes do ***Workspace*** por meio de uma única transmissão.

De acordo com o Modelo de Título de Entidade, ***Workspace*** pode ser definido como um barramento lógico que independe de topologias e natureza da infra estrutura de redes interconectadas.

A associação da identificação por Título com o conceito de ***Workspace*** viabiliza a propriedade de mobilidade. Uma Entidade pode se 'deslocar' ao longo do barramento que materializa o ***Workspace*** sem prejuízo para a qualidade de serviço ajustada para aquele domínio de comunicação. Além da mobilidade, o conceito de ***Workspace*** permite uma outra propriedade essencial nos dias atuais relativamente a segurança, uma vez que é intrinsecamente confidencial.

2.3 Arquitetura ETArch

O conceito de comunicação por difusão seletiva (*Multicasting*) está presente desde as primeiras redes. No entanto, devido a questões técnicas, esse tipo de comunicação ainda apresenta diversos desafios e deficiências dependendo da rede que o implementa. Difusão seletiva, mobilidade, qualidade de serviço e qualidade de experiência estão entre os requisitos de comunicação que não são completamente suportados pela Internet. O principal objetivo da Arquitetura de Título de Entidade é implementar a aproximação semântica vislumbrada pelo Modelo de Título de Entidade, afim de habilitar experimentos que constatem que a aproximação semântica é um caminho promissor no desenvolvimento de uma rede mais aderente às necessidades de comunicação.

Aspectos Conceituais da ETArch

2.3.1 Serviço de Domínio de Títulos

ETArch é uma arquitetura baseada no Modelo de Título de Entidade, que mantém logicamente centralizada a inteligência da rede. O sistema distribuído responsável por manter a inteligência da rede é denominado DTS (*Domain Title Service*). Posto de outra forma, o DTS é um sistema distribuído que implementa o plano de controle da arquitetura ETArch e mantém a inteligência da rede distribuída através de agentes interconectados denominados DTSA.

O DTS mantém e controla todos os aspectos ligados (i) à resolução de Títulos, (ii) ao gerenciamento do ciclo de vida de Entidades no ambiente distribuído e, particularmente, (iii) a manutenção da relação entre Entidades (seção 2.2.2 e Títulos (seção 2.2.1). Como o plano de dados, através do qual as Entidades se comunicam, é materializado através de *Workspaces* (seção 2.2.3), então, o DTS também é responsável pelo ciclo de vida de *Workspaces*.

Em termos de *Workspaces*, a ETArch especifica dois tipos: (i) *Workspace* de Controle, que é criado a priore, em tempo de *bootstrap* da rede, e é responsável pela conectividade lógica de DTSAs, pelo qual circulam apenas primitivas de serviços do plano de controle; e (ii) *Workspace* de Dados, que é criado sob demanda, por Entidades desejosas de comunicação (com outras Entidades), pelo qual circulam primitivas de dados relativos aos protocolos das entidades de aplicação do plano de dados. O DTS mantém a base de conhecimentos sobre os detalhes do ambiente (*Workspaces* de Controle, *Workspaces* de Dados, Entidades, Títulos e Elementos de Rede) e, portanto, é capaz de monitorar e controlar os requisitos de comunicação de Entidades ao longo do tempo.

O DTS é composto por um conjunto de um ou mais DTSAs, como representado na Figura 1, os quais são responsáveis por implementar aspectos da gestão de uma localidade (sub conjunto) do domínio. A interconexão de DTSAs é feita por Elementos de Redes (NE – *Network Element*), por exemplo *switches*, que podem ser exclusivos para o Plano de Controle (*Control Plane*) ou compartilhados com o Plano de Dados (*Data Plane*).

A responsabilidade pela manutenção da conectividade lógica da infraestrutura, da mesma forma, por exemplo, na presença de falha em elementos de rede, faz com que o DTS seja capaz de realizar a reconfiguração dos *Workspace* afetados, a fim de manter a comunicação.

Os NEs representados na Figura 1 constituem o plano de dados da arquitetura ETArch e são monitorados/controlados pelo DTS (de fato pelos DTSAs) para que atendam aos requisitos de comunicação das Entidades (Aplicações). O controle/monitoração dos NEs é feita em regime permanente e, portanto, podem modificar seus comportamentos em tempo de execução.

Essa abordagem, proposta inicialmente no Modelo de Título de Entidade, encaixa-se naturalmente às abstrações de SDN, onde um controlador centralizado é capaz de alterar o comportamento do plano de dados.

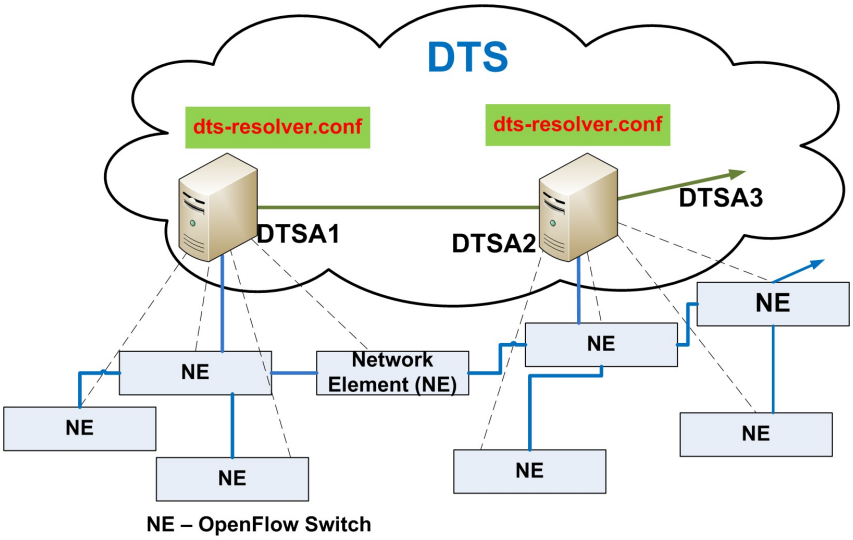

Figura 1 – Visão Geral do DTS

2.3.2 Camadas ETArch

O padrão arquitetural em Camadas (*Layer*) é adotado como ponto de partida para a especificação da Arquitetura ETArch. Embora utilize uma abordagem *clean slate*, ETArch se propõe a suportar protocolos da camada de Aplicação e a utilizar protocolos da camada Física existentes, levando-se em conta a nomenclatura utilizada pelo modelo de referência OSI (ZIMMERMANN, 1980).

Como a ETArch é uma arquitetura que separa os planos de Controle e de Dados, então há que imaginar que haverá uma pilha de protocolos para cada um dos planos mencionados, conforme ilustra a Figura 2. Observe que a arquitetura do Plano de Controle é uma estrutura 'rígida' projetada para suportar os requisitos de controle da infraestrutura, que são especificados por um *Metadata* bem definido. Por outro lado, o Plano de Dados (na figura apenas *Data*) é uma estrutura dinâmica e depende da aplicação (na figura representada pela Entidade na *Userland*) que no momento faz uso da infraestrutura de comunicação.

Nos dias atuais, estas camadas que são identificadas por uma configuração dinâmica (*Dynamic Layer Setup*), remetem à estrutura de camada rígidas que podem ser encontradas em diversas arquiteturas, como por exemplo, a arquitetura Internet. Além de rígidas, essas camadas introduzem *overhead* significativo ao caminho dos dados. A arquitetura ETArch redesenha esta estrutura e define, de cima para baixo, três camadas denominadas: *Application*, *Communication* e *Link*, representadas na Figura 3 a seguir. No geral a arquitetura ETArch reexamina as camadas de Transporte e de Rede da arquitetura Internet e do modelo de referência OSI. Com esta premissa, o foco deste trabalho é basicamente a camada *Communication*.

A representação não usual da camada *Communication* tem o objetivo de transmitir a capacidade que a arquitetura ETArch tem de se adaptar a requisitos de Entidades de

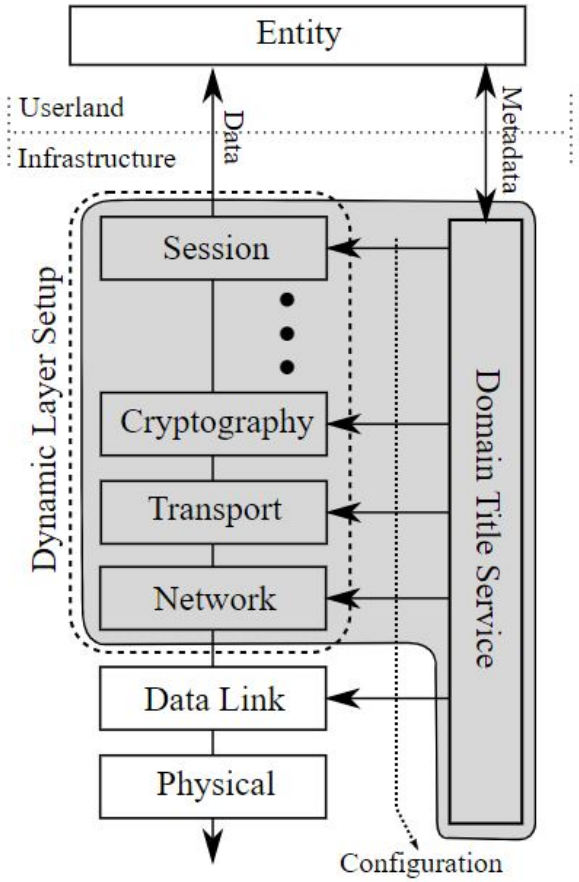

Figura 2 – Visão Global dos aspectos de Controle e de Dados

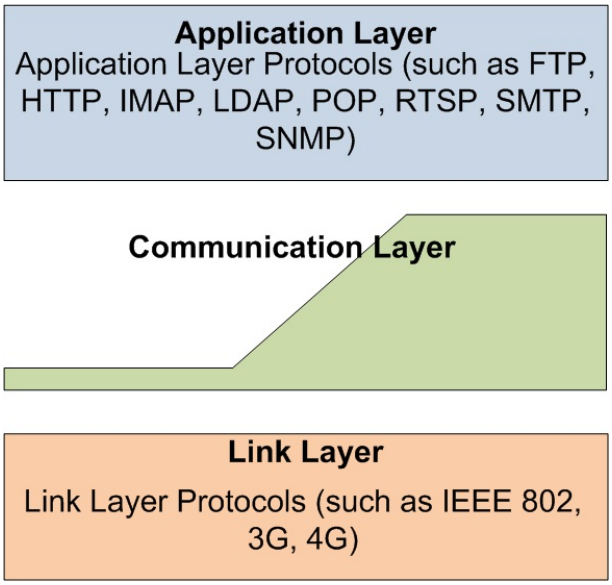

Figura 3 – Camadas definidas pela ETArch

Aplicação de forma dinâmica. A configuração sob demanda das camadas da Figura 3 é feita pelo plano de Controle representado pelo DTS (*Metadata*) na Figura 2. Dependendo do contexto de comunicação, pode ser que o *overhead* da camada *Communication* seja mínimo, representado pela porção delgada (lado esquerdo da Figura 3), ou até mesmo inexistente. Por exemplo, imagine que duas Entidades de Aplicação estejam se comunicando em uma rede local através de serviços não orientados a conexão (*ConnectionLess Services*), que não ofereçam garantia de entrega, então elas precisariam apenas do enlace lógico e a camada de Rede seria desnecessária.

No outro extremo, pode ser que o contexto de comunicação necessite de funcionalidades que imponham *overhead* máximo da camada *Communication*, representado pelo lado direito da Figura 3. Suponha-se, por exemplo, que duas ou mais Entidades de Aplicação estejam distribuídas em diversos domínios (redes) e se comuniquem através de serviços orientados a conexão (*Connection Oriented Services*), com garantia de entrega e requisitos de segurança (confidencialidade, integridade, não repúdio, autenticidade, entre outros), então elas precisariam de roteamento, serviços confirmados, criptografia, certificados etc.

No plano de Controle, o compromisso é fazer com que a configuração de *Workspaces* de Dados atendam a requisitos de QoS ou QoE definidos por Entidades de Aplicação (*Userland*). Observe-se que neste plano há a necessidade de interações com NEs e, então, a filosofia SDN vem ao encontro do que a ETArch se propõe. O DTS (de facto DTSAs), através de *Workspaces* de Controle, tem a responsabilidade de compreender os requisitos de Entidades de Aplicação e traduzi-los para a configuração de NEs na criação ou extensão de *Workspaces* de Dados.

Em relação à camada *Link* subjacente, a adoção da filosofia SDN permite que a camada *Communication* utilize interfaces com serviços de enlace padronizados (*South Bound Interface* (SBI)), o que a torna naturalmente compatível com os serviços atualmente oferecidos pelas tecnologias de enlace atuais, como *OpenFlow*, e emergentes.

O plano de Controle tem a missão de abstrair os detalhes da configuração de NEs, de modo que os requisitos de Entidades de Aplicação sejam perfeitamente compreendidos e atendidos na construção de *Workspaces* de Dados. Desse modo, os serviços de *Workspaces* de Dados são reduzidos em número, mas com classes de abstração conforme mostra a seção 2.3.3.

2.3.3 Serviços ETArch

No plano de Dados, como o próprio nome diz, o compromisso é com a transmissão de primitivas de dados, que são de responsabilidade das Entidades de Aplicação (*Userland*). Para suportar a natureza dinâmica de requisitos das entidades de aplicação, ETArch oferece basicamente três classes de serviços:

- **Serviços Confirmados**: serviço no qual a entidade requisitante do serviço de

dados recebe uma confirmação (*Acknowledgement* ou simplesmente *Ack*) para cada requisição;

- **Serviços Não-confirmados**: serviço no qual a entidade requisitante do serviço de dados não recebe evidência de que a primitiva foi entregue; e,

- **Serviços *Piggybacking***: serviço no qual a entidade requisitante do serviço de dados, ao mesmo tempo em que requisita um envio de dados aproveita para confirmar o recebimento de dados recebidos por ela anteriormente.

Observe que os Serviços Confirmados (*Confirmed Services*) servem a entidades que requerem uma evidência de que a primitiva atingiu seu(s) destino(s) e, portanto, são comumente referenciados como serviços confiáveis, enquanto os Serviços Não-confirmados (*Unconfirmed Services*) são por natureza não confiáveis uma vez que a Entidade enviadora não tem uma evidência de que a primitiva foi entregue. Note-se que o termo "Confiável" tem o significado de que a Entidade requisitante sabe o que aconteceu com a primitiva (chegou sem erro, chegou com erro ou não chegou).

Serviços *Piggybacking* são serviços confiáveis, uma vez que a entidade requisitante recebe uma evidência da chegada dos dados ao destino, mas é interessante de ser utilizado em relações balanceadas nas quais as entidades participantes requisitam primitivas de dados nas mesmas quantidades. Nessas condições, este tipo de serviço apresenta a mesma funcionalidade dos Serviços Confirmados, mas ocupando bem menos recursos de comunicação (vazão, banda de comunicação, memória e processamento) e menor probabilidade de erro.

2.3.4 Identificação e Endereçamento

Identificação e Endereçamento são dois conceitos centrais para as comunicações entre entidades pares em um ambiente distribuído. Basicamente, as primitivas de dados são enviadas e recebidas por Entidades, sendo que seus Títulos (identificações) indicam "quem" são esses pares e os respectivos endereços indicam "onde" estão e como podem ser alcançadas (as entidades pares).

O Título é responsável por designar a existência de uma entidade e permitir sua identificação de forma única, não ambígua e independente de topologia. O endereço, por sua vez, indica "onde" essa entidade está localizada e de posse dessa localização (SALTZER, 1993) permite construir um caminho para a comunicação entre essas entidades pares.

Na ETArch, existe uma separação entre Títulos e endereços. Títulos identificam precisamente uma entidade de forma independente de sua localização. Os endereços por sua vez são dependentes da localização. Se uma entidade se move, seu título não se altera, mas o endereço de seu *Point of Attachment* (PoA) pode ser alterado.

Conforme definido na seção 2.2.2, Entidades trocam interações via *Workspace* no qual cada uma é identificada unicamente por seus respectivos Títulos. Considerando que existem entidades em todas as camadas da arquitetura e que as entidades de aplicação fazem uso dos serviços residentes em camadas subjacentes, então pode ser que o título de uma entidade de aplicação se derive de títulos associados a entidades subjacentes.

Por exemplo, do ponto de vista da ETArch, um *host* ou *smartphone* são Entidades e, portanto, possuem Títulos, então, programas ou sensores residentes nesses dispositivos podem ter um título único que é gerado a partir do título do dispositivo no qual está hospedado. Observe-se que essa abordagem não fere o princípio da independência de localização do título, pois essas camadas podem ensejar aspectos da implementação e da arquitetura da máquina, mas não fixam a localização (SILVA et al., 2012) e (SANTOS et al., 2011).

Capítulo 3

Modelagem Conceitual Orientada a Ontologia

Os modelos conceituais foram introduzidos para aumentar o entendimento e a comunicação entre as partes interessadas. Um modelo conceitual possui três características: (1) **mapeamento**, significando que o modelo é uma representação do sistema original, expresso através de uma linguagem de modelagem; (2) **redução**, caracterizando o modelo como um subconjunto do sistema original; e (3) **pragmática**, que descreve o propósito ou objetivo pretendido. Modelagem conceitual é a atividade de representar aspectos do mundo físico e social com o propósito de melhorar a comunicação, o aprendizado e a solução de problemas.

As ontologias se mostraram bastante úteis em medir se diferentes procedimentos de modelagem conceitual podem levar a boas representações de fenômenos do mundo real, sendo, portanto, rapidamente adicionadas à área de modelagem conceitual, inicialmente como forma de avaliar a correição ontológica de linguagens de modelagem conceitual (ex.: UML, ORM, ER, REA, OWL)[1]. Além disso, ontologias podem expressar os elementos fundamentais de um domínio e tornar-se a fundação teórica de um modelo conceitual.

Nesse sentido, a modelagem conceitual orientada a ontologia pode ser definida como a utilização de teorias ontológicas para desenvolver artefatos de engenharia (por exemplo, linguagens de modelagem, metodologias, padrões de projeto e simuladores) para melhorar a teoria e a prática da modelagem conceitual. Assim, todas essas técnicas têm em comum o uso de ontologia (avaliação, análise, fundamentação teórica ou interoperabilidade) para melhorar o mapeamento, a redução ou a pragmática de processos de modelagem conceitual, ou de seus produtos, os modelos conceituais (VERDONCK et al., 2015).

[1] UML – *Unified Modeling Language*, ORM – *Object Relational Mapping*, ER – *Entity Relationship*, REA – *Resources Events and Agents*, OWL – *Ontology Web Language*

3.1 Definição de Ontologia

Uma conceitualização é uma visão abstrata e simplificada do mundo que criamos com algum propósito. Toda base de conhecimento, sistema ou agente relacionado a essa base está comprometido com alguma conceitualização. A despeito da complexa noção psicológica de "conceitualização", ela pode ser explicada usando uma representação matemática bem simples, chamada estrutura relacional extensional. Uma conceitualização ou estrutura relacional extensional é uma tripla $C = (D, W, R)$, onde D é um conjunto de elementos que compõe o universo de Discurso, ou seja, um conjunto de objetos que existem no mundo que queremos conceitualizar, W é o conjunto de possíveis Mundos (*World*), e R é um conjunto de relações conceituais no espaço de domínio $< D, W >$ (GUARINO; OBERLE; STAAB, 2009).

Suponha uma rede com mil equipamentos, cada um identificado com seu número de série (identificador), por exemplo. Vamos assumir que o universo de discurso D contenha todos esses identificadores e que o interesse se resume às relações entre esses equipamentos. O conjunto R conterá algumas relações como *Equipment*, *Host*, *Switch*, *Router*, e também algumas relações binárias como *linked-with* e *sent-packet-to*. A estrutura de relacional se parece com o seguinte:

- D = {ZAQ9786, CFD5340, ABX3045, TWB7198, ACT2864, DKC3762, ...}
- W = {w1, w2, ...}
- R = {Equipment, Host, Switch, Router, linked-with, sent-packet-to}

As relações extensionais refletem um mundo específico. O universo de discurso é o conjunto D e *Host*, *Switch* e *Router* são relações que se aplicam apenas a subconjuntos de D. A relação binária *linked-with* é um conjunto de tuplas que especificam cada ligação que um equipamento mantém com os outros.

Para manter a simplicidade vamos assumir que as relações unárias são as mesmas em todos os mundos, então para todos os w em W:

- Equipment(w) = D
- Host(w) = {..., ZAQ9786, DKC3762, ...}
- Switch(w) = {..., TWB7198, ACT2864, ...}
- Router(w) = {..., CFD5340, ABX3045, SDJ1439, ...}

Mas as relações binárias são diferentes:

- linked-with(w1) = {..., (ACT2864, CFD5340), (DKC3762, ACT2864), ...}

- linked-with(w2) = {..., (TWB7198, ABX3045), (ZAQ9786, TWB7198), ...}
- sent-packet-to(w1) = {..., (DKC3762, ACT2864), ...}
- sent-packet-to(w2) = {..., (ZAQ9786, TWB7198), ...}

A comunicação, seja ela entre máquinas ou humanos, exige uma linguagem para se referir aos elementos da conceitualização. Por exemplo, para expressar o fato de que o DKC3762 está conectado ao ACT2864 precisamos introduzir um símbolo específico, ou posto de outro modo, formalmente um símbolo na função de predicado, digamos *linked-with*, que no entendimento dos usuários, deve representar uma certa relação conceitual. Digamos, neste caso, que nossa linguagem L se compromete com determinada conceitualização. Suponha agora que L é uma linguagem lógica com o seguinte vocabulário { Host, Switch, Router, linked-with, sent-packet-to}. Como podemos ter certeza que serão interpretados de acordo com a conceitualização que a linguagem L se compromete? Como garantir que alguém que não seja da área de redes interprete *linked-with* como uma conceitualização *send-packet-to* e vice versa? O significado pretendido com o predicado *linked-with* será definido pelo retorno de sua interpretação contendo uma das possíveis extensões para cada mundo possível da relação conceitual denotada pelo predicado. O problema é que para especificar essas extensões precisamos especificar nossa conceitualização de forma explícita, mas conceitualização é um processo que ocorre na mente das pessoas, sendo tipicamente implícito.

A ontologia surge então como uma especificação implícita de uma conceitualização. Podemos especificar uma conceitualização, de forma extensional ou intensional. A especificação extensional exige que todos os estados de todos os possíveis mundos sejam descritos, o que pode se tornar impossível ou impraticável. Outra maneira de especificar uma conceitualização é criar uma linguagem que será usada para conversar sobre ela, e restringir as interpretações dessa linguagem de forma intensional aplicando axiomas a cada predicado. A ontologia, portanto, é o conjunto de fórmulas intensionais, isto é, descrições através dos aspectos mais essenciais associadas a símbolos, de forma que quando um objeto é mencionado, é possível decidir se o objeto pode ou não ser representado por determinado símbolo. Por exemplo, pode-se dizer que *linked-with* é uma relação entre dois elementos do conjunto *Equipment* e que essa relação possui as propriedades simétrica e transitiva, enquanto que *send-packet-to* também é uma relação entre dois elementos do conjunto *Equipment*, mas possui as propriedades assimétrica e intransitiva.

Relações extensionais, que citam exemplos ou usam propriedades que permitem identificar prontamente os objetos em questão são ideais para os seres humanos, pois através da observação eles conseguem extrair e classificar os objetos por suas características essenciais e aos poucos constroem a sua conceitualização. No entanto, a utilidade das máquinas de inferência reside na sua capacidade de dar respostas e tomar decisões em uma escala de tempo desprezível em relação ao tempo necessário para construir uma conceitualização.

Considerando que as conceitualizações já existem, então, é necessário representá-las e garantir que elas serão interpretadas de acordo com a realidade que se quer abstrair. Assim, para uma dada linguagem L com vocabulário V e as relações extensionais de determinado domínio $S = (D, R)$, a semântica de V é dada pelo modelo $M = (S, I)$, onde I é função de interpretação extensional $I : V \rightarrow D \cup R$. Assim, a semântica de V também pode ser dada pelo compromisso ontológico $K = (C, I)$, onde I é a função de interpretação intensional $I : V \rightarrow D \cup R$ (GUARINO; OBERLE; STAAB, 2009).

Assim, assuma uma conceitualização C, uma linguagem lógica L com um vocabulário V e um comprometimento ontológico K. Uma ontologia O_K que garante o comprometimento ontológico K para uma conceitualização C com vocabulário V é uma teoria lógica composta por um conjunto de fórmulas de L. Esse conjunto de fórmulas pode conter, por exemplo:

- **Informação taxonômica**: usada para especificar que algumas relações unárias são elementos de outras relações unárias, construindo dessa forma uma hierarquia de relações unárias.

  $O_1 = \{Host(x) \rightarrow NetworkElement(x), Switch(x) \rightarrow NetworkElement(x), Router(x) \rightarrow NetworkElement(x)\}$

- **Domínios e Faixas**: uma relação é uma tupla ordenada de elementos, de forma que cada posição da tupla implica em diferentes tipos de elementos.

  $O_2 = \{\text{linked-with(x,y)} \rightarrow \text{NetworkElement(x)} \wedge \text{NetworkElement(y)}\}$

- **Simetria**: algumas relações que se estabelecem na ordem direta passam também a valer imediatamente em ordem inversa.

  $O_3 = \{\text{linked-with(x,y)} \leftrightarrow \text{linked-with(y,x)}\}$

- **Transitividade**: Algumas relações encadeadas passam a ser aplicadas diretamente do primeiro ao último elemento da cadeia.

  $O_4 = \{\text{linked-with(x,z)} \leftarrow \text{linked-with(x,y)} \wedge \text{linked-with(y,z)}\}$

  $O_k = O_1 \cup O_2 \cup O_3 \cup O_4$

3.2 Unified Foundational Ontologies

Ontologias de Fundamentação (*Foundational Ontologies*) são sistemas de categorias filosoficamente bem fundamentados e independentes de domínio, utilizados com sucesso

para melhorar a qualidade de linguagens de modelagem e modelos conceituais. Visando obter uma linguagem de modelagem de alta qualidade, foi tomado como base o sistema de categorias bem fundamentadas das Ontologias de Fundamentação Unificadas (UFO – *Unified Foundational Ontologies*). UFO permite produzir um sistema de categorias específico para cada linguagem e isso equivale a dizer que cada linguagem terá um sistema próprio de classes e regras para classificação de elementos do vocabulário nessas classes específico para cada linguagem. Um sistema específico permite produzir linguagens de modelagem otimizadas para os domínios e situações em que são utilizadas. A ontologia de objetos da UFO-A, provê algumas categorias mostradas na figura 4, que podem ser usados na construção de taxonomias específicas. Com o intuito de esclarecer como essa ferramenta funciona, alguns exemplos são tratados a seguir (GUIZZARDI, 2005).

Substanciais («Substantial») são universais («Universal») existencialmente independentes. Exemplos incluem objetos mesoscópicos do senso comum, tais como uma pessoa, um cachorro, uma casa, Tom Jobim e Os Beatles. A palavra modo («Mode»), em contraste, denota a instanciação de uma propriedade. Um modo é um universal que só pode existir em outros indivíduos e é dito ser inerente a esses indivíduos. Exemplos típicos de Modos são uma cor, uma carga elétrica, um sintoma etc. Um importante traço que caracteriza todos os Modos é o fato deles só poderem existir em outros indivíduos. Por exemplo, uma carga elétrica só pode existir em algum condutor.

Colocando de forma mais técnica, diz-se que um Modo é existencialmente dependente de outros indivíduos. A dependência existencial também pode ser usada para diferenciar Modos Intrínseco («Intrinsic Mode») e Relacional («Relator»). Modos Intrínsecos são dependentes de um único indivíduo (por exemplo uma cor, uma dor de cabeça, uma temperatura etc.) e Modo Relacional («Relator») depende de vários indivíduos (por exemplo um emprego, um tratamento médico, um casamento, etc.). Substanciais («Substantial») são universais existencialmente independentes e Modo («Mode») são universais que dependem existencialmente de algum substancial. Exemplos do primeiro incluem Maçã, Planeta e Pessoa. Exemplos do último incluem Cor, Carga Elétrica e Dor de Cabeça.

Relações são entidades que aglutinam outras entidades. Na literatura de Filosofia, duas categorias amplas de relações são tipicamente consideradas, a saber relações Formais e Materiais. Relações Formais acontecem entre duas ou mais entidades diretamente, sem a interferência de nenhum outro indivíduo. Em princípio, a categoria de Relações Formais inclui aquelas relações que formam a superestrutura matemática de nosso arcabouço, incluindo dependência existencial, tais como *parte-de*, *subconjunto-de*, *instanciação*, dentre outras. Relações Materiais, por outro lado, possuem estrutura material por si próprias e incluem exemplos como *trabalhar-em*, *estar-matriculado-em* ou *estar-conectado-a*.

Por exemplo, enquanto, a Relação Formal entre *Paulo* e seu *conhecimento x de Grego* acontece diretamente, tão logo *Paulo* e x existam. A Relação Material entre *Paulo* e uma unidade médica H, requer a existência de uma outra entidade para mediar *Paulo*

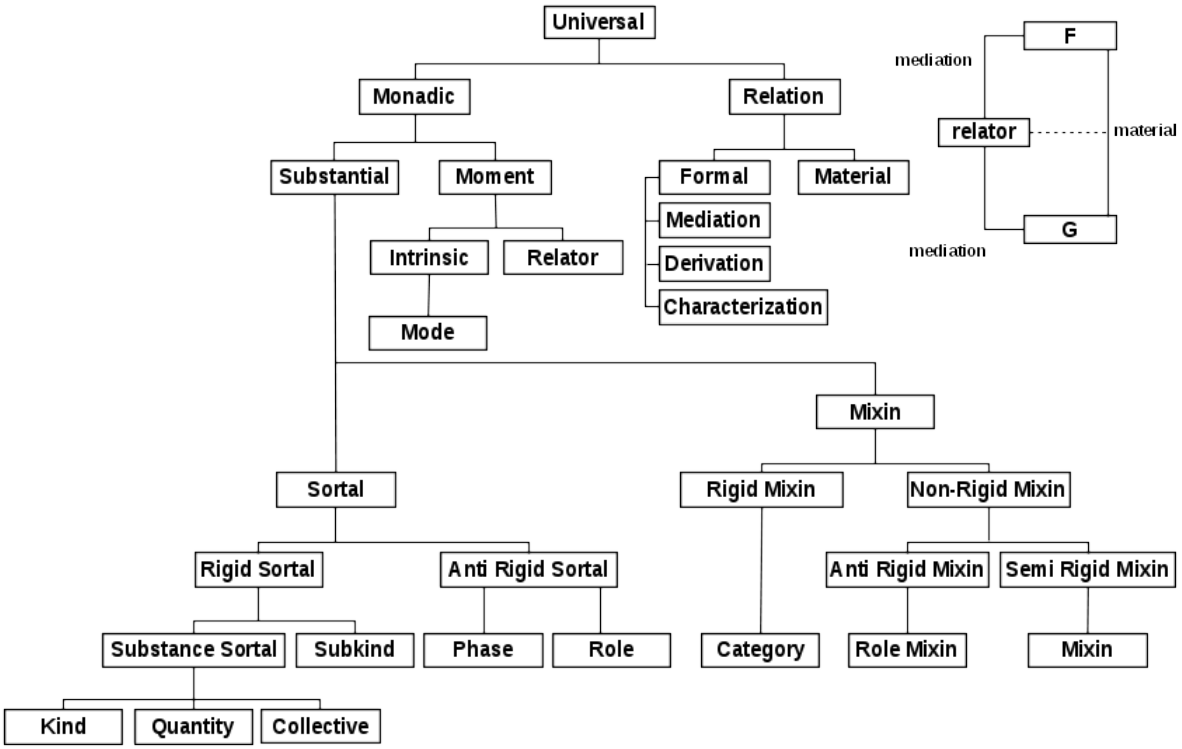

Figura 4 – Taxonomia de Universais (GUIZZARDI, 2005)

e H, neste caso um tratamento. Essas entidades são denominadas Modos Relacionais («Relators»), que são indivíduos com o poder de conectar (mediar) outros indivíduos. Assim, um tratamento médico conecta um paciente a uma unidade médica, uma matrícula conecta um estudante a uma instituição de ensino etc.

Suponha que João está casado com Maria. Neste caso, assume-se que há um modo relacional do tipo casamento que media João e Maria. Além disso, há várias propriedades que João adquire em virtude de estar casado com Maria, bem como responsabilidades legais que João assume no contexto dessa relação. Essas novas propriedades adquiridas são instanciadas em modos intrínsecos de João que, por conseguinte, são existencialmente dependentes dele. Entretanto, esses modos também dependem da existência de Maria. Esse tipo de modo é chamado de modo externamente dependente, i.e, modos intrínsecos que são inerentes a um único indivíduo, mas que são existencialmente dependentes de outros (possivelmente vários) indivíduos. O modo relacional *casamento*, neste caso, é a soma de todos os modos externamente dependentes que João e Maria adquirem em virtude de estarem casados (GUIZZARDI; FALBO; GUIZZARDI, 2008).

O processo de classificação de universais do universo de discurso é realizado com base na teoria ontológica, chamada *Unified Foundational Ontologies*. As principais categorias e regras de classificação são descritas na tese de doutorado *Ontological Foundations for Structural Conceptual Models* (GUIZZARDI, 2005). Note que na raiz da árvore temos o Universal, representando todos os predicados. Esses predicados são divididos em dois tipos: *Monadic*, atribuídos a uma entidade única; e *Relation*, atribuídos a pelos menos

duas entidades. *Monadic Universals* se dividem em *Substantial* e *Moment*. *Sortal* são universais que carregam um princípio de identidade. As interações entre esses *Universals* é regida por relações e restrições descritas na própria teoria ontológica.

3.3 OntoUML

OntoUML (GUERSON et al., 2015) é uma linguagem para Modelagem Conceitual Orientada a Ontologia, construída como uma extensão à UML (*UML Profile*) com base na Ontologia de Fundamentação Unificada (UFO). A OntoUML é usada para expressar modelos ontológicos, que são representados por meio de diversos diagramas. Modelos capturam a essência de alguns aspectos de determinada realidade, que pode ser um sistema ou um domínio de conhecimento. Nessa tese, a OntoUML é usada como uma linguagem de especificação para representação de conhecimento. A OntoUML está alicerçada na distinção ontológica entre Tipos e Indivíduos.

Tipos são construções abstratas, criados para auxiliar na percepção e classificação do mundo que nos cerca, e representam conjuntos de características que se espera encontrar na realidade, que está repleta de objetos que exemplificam esses conjuntos. Objetos são instâncias de tipos e são denominados Indivíduos. Note que não há como especificar Indivíduos em OntoUML. Um modelo OntoUML é composto por diagramas, e um diagrama pode conter vários Tipos. Um Tipo é construído com classes e relacionamentos (ou associações), aplicando as noções ontológicas de identidade, rigidez e dependência aos conceitos de determinado domínio.

são especificados por um nome, um estereótipo e um símbolo gráfico. De acordo com as regras sintáticas da linguagem OntoUML, Tipos complexos também pode ser especificados, os quais podem conter vários nomes, estereótipos e símbolos gráficos.

3.3.1 Classes

A Figura 5 ilustra a definição de dois tipos. O nome do primeiro tipo é **Pessoa** e o nome do segundo é **Switch**, sendo que ambos usam o estereótipo «Kind» e o símbolo gráfico que representa classe (retângulo). No processo de definição de tipos, deve-se sempre considerar as noções ontológicas de Identidade, Rigidez e Dependência. Por ora, o modelo OntoUML tem apenas um diagrama, sendo que esse diagrama tem apenas dois tipos definidos: **Pessoa** e **Switch**. O tipo **Pessoa** representa indivíduos como você e eu, em outras palavras, pode-se dizer que você e eu somos instâncias do tipo **Pessoa**. Assim, pode-se igualmente dizer que os *switches*, que estão na sua e na minha rede, são instâncias do tipo **Switch**. Dessa forma, ao visualizar os tipos **Pessoa** e **Switch**, que estejam em qualquer outro diagrama, sabe-se a quais instâncias eles podem ser aplicados. Como o domínio do modelo não foi delimitado e os tipos definidos são simples, pode-se dizer que qualquer “pessoa” e qualquer “switch” são instâncias desses respectivos tipos.

Figura 5 – Classes OntoUML

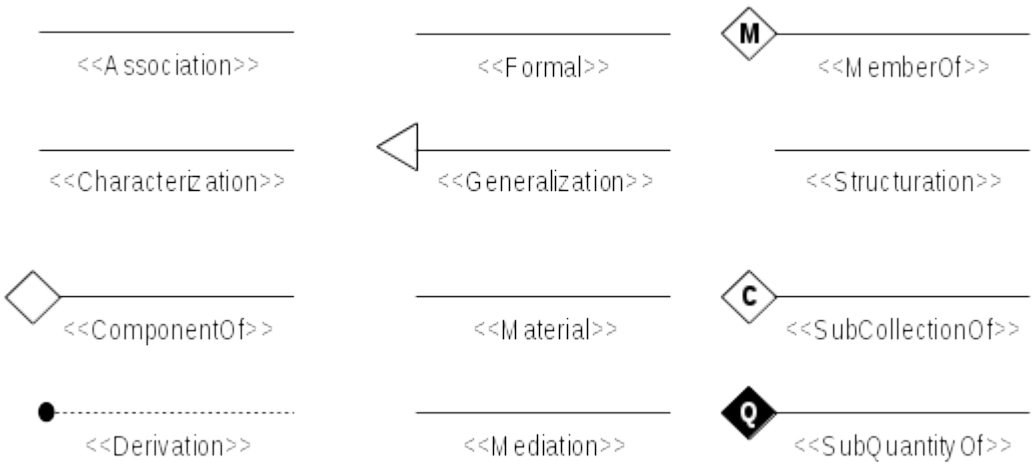

Figura 6 – Relacionamentos OntoUML

Embora esse modelo não possua muita utilidade, seu objetivo é demonstrar que o processo de modelagem é incremental e os detalhes são acrescentados à medida que o entendimento sobre o domínio aumenta. É importante lembrar que Tipos podem ser agrupados para definir outros tipos mais complexos.

3.3.2 Relacionamentos

Os símbolos gráficos usados nos tipos relacionamento são linhas usadas nos diagramas para associar os retângulos conhecidos como classes. Os tipos de relacionamento são diferenciados estereótipos de relacionamento («Generalization» e «Mediation», por exemplo.), em alguns casos a distinção pode ser feita por detalhes gráficos em uma das extremidades, conforme descrito na Figura 6, nos quais o estereótipo pode ser omitido.

Um relacionamento estereotipado como «Formal» é usado para representar relações que podem ser reduzidas a comparações de valores que caracterizam os indivíduos relacionados como mais-pesado-que, mais-jovem-que ou mais-barato-que.

Relação «Material» tem uma estrutura material e incluem exemplos como empregos, beijos, matrículas, conexões de voo e compromissos. Os indivíduos relacionados por uma relação «Material» são mediados por outros indivíduos estereotipados como «Relator». *Relators* são indivíduos com o poder de conectar entidades. Por exemplo, uma Conexão de voo é um *relator* que conecta aeroportos, e uma Matrícula é um *relator* que conecta um

estudante a uma instituição de ensino. *Relators* desempenham um importante papel em responder questões do tipo: O que significa dizer que João é casado com Maria? Porque é certo dizer que Pedro trabalha para a empresa X e não para a Y?

Relações do tipo «Material» são derivadas (via «Derivation») de *Relators* e de relações de mediação que os conectam a indivíduos relacionados. As restrições de cardinalidade das relações de mediação colapsam por derivação. Relações do tipo «Material» sempre são afetadas por colapsos de cardinalidade. Além disso, diversas relações «Material» podem ser derivadas de uma única relação entre um «Relator» e um «Mediation».

«Mediation» é uma relação entre um «Relator» e as entidades que ele conecta. Uma Mediação pode ser entendida como um tipo de relação de Dependência Existencial. Ela pode ser derivada da relação entre os indivíduos relacionados e os *qua individuals* (MASOLO et al., 2005) que compõem o «Relator» e são inerentes aos indivíduos relacionados. Um «Relator» deve mediar ao menos dois indivíduos distintos.

«Characterization» é uma relação entre um Tipo Portador e sua Característica. Uma Característica é um momento (inerente) intrínseco do portador e, portanto, existencialmente dependente desse tipo. Características podem ser estereotipadas como «Quality» ou «Mode». Uma Característica diferencia um Tipo Portador se e somente se todas as instâncias do portador apresentam a característica.

3.3.3 Identidade

Identidade é uma noção ontológica importante na definição de tipos complexos, uma vez que todos os indivíduos devem possuir uma identidade, mas nem todos os tipos a fornecem. Considere um indivíduo que instancia o tipo Pessoa e que ao longo do tempo perdeu peso. Aquele indivíduo que se conheceu há algum tempo e o indivíduo que se vê agora são o mesmo ou são indivíduos distintos? Durante o processo de emagrecimento, aquele indivíduo (gordo) foi destruído e outro (magro) foi criado para substituí-lo? Os seres humanos definem naturalmente um princípio de identidade para todos os tipos que conhecem. Esse princípio lhes permite afirmar que, nesse caso, o indivíduo gordo e o magro são o mesmo e, de fato, humanos podem confirmar a identidade de pessoas em circunstancias muito mais complexas.

No entanto, computadores necessitam de regras bem definidas e formais para realizar distinções semelhantes. Assim, o uso da linguagem OntoUML nesse método de modelagem se justifica pelo fato da linguagem estar apoiada em uma teoria ontológica que provê essas regras. O princípio de Identidade é como uma função que deve retornar sempre o mesmo indivíduo, contanto que alguma circunstância relacionada a esse indivíduo seja usada como entrada. Adicionalmente, a função deve ser válida para todos os indivíduos de determinado tipo.

Assim, divide-se os Tipos em três grupos: i) os que fornecem identidade; ii) os que precisam de identidade; e iii) aqueles que nem fornecem e nem precisam de identidade.

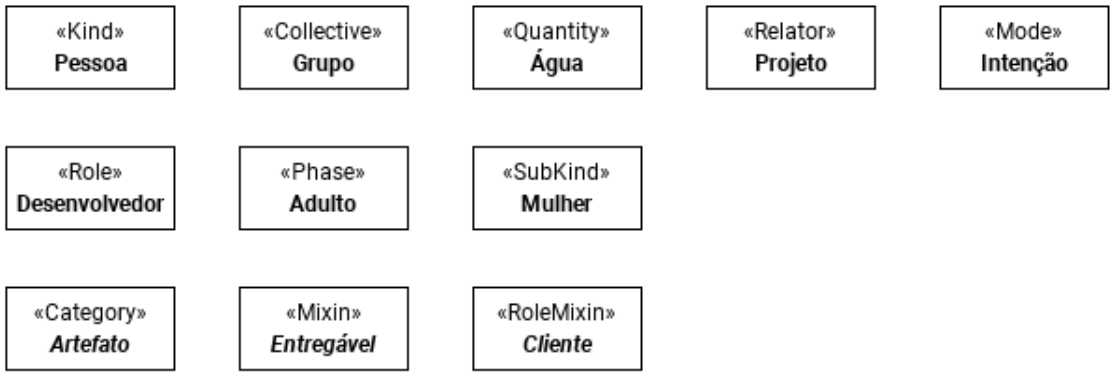

Figura 7 – Exemplos de Tipos: Provedores, Consumidores e Sem Identidade

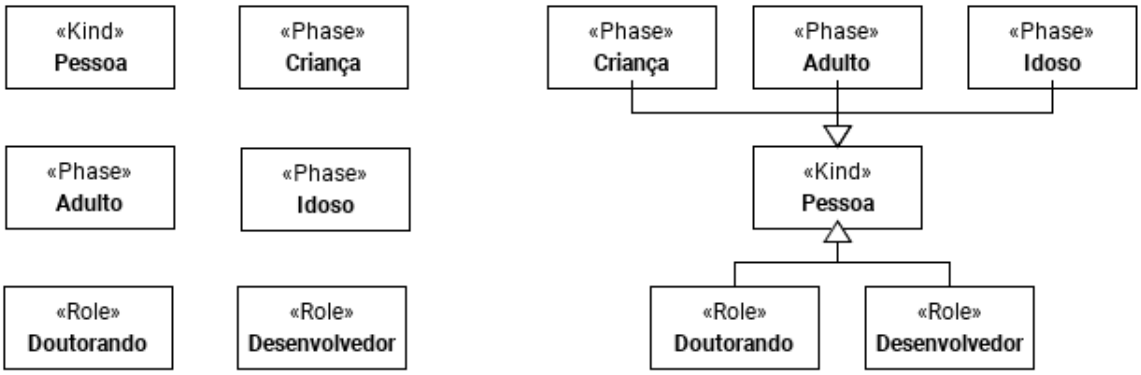

Figura 8 – Generalização por identidade

Aqueles (i) que proveem identidade, devem ser estereotipados como «Kind», «Collective», «Quantity», «Relator» ou «Mode». Aqueles (ii) que precisam de identidade, herdam e compartilham a identidade de um tipo do grupo (i), e devem ser estereotipados como «Subkind», «Role» ou «Phase». As instâncias de tipos do grupo (iii) que não proveem e nem necessitam de identidade, devem obedecer princípios de identidade de diferentes tipos dos grupos (i) e (ii), e devem ser estereotipados como «Category», «Mixin» ou «RoleMixin». A Figura 7 mostra exemplos desses grupos.

3.3.4 Rigidez

Rigidez é outra noção ontológica importante na definição de Tipos complexos, uma vez que nem todos os tipos contidos na definição de um tipo complexo estão instanciados durante toda a existência de um indivíduo complexo. Considere um indivíduo João que instancia o tipo Pessoa. Há muitos anos atrás João era uma criança, no presente é um adulto e espera-se que um dia seja um idoso. Durante toda a sua existência, João instancia o tipo Pessoa, mas instanciou durante parte de sua vida o tipo Criança. Atualmente,

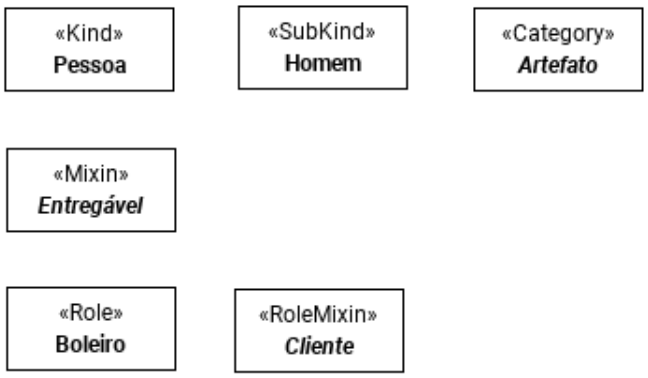

Figura 9 – Exemplos de tipos Rígidos, Semi-Rígidos, e Anti-Rígidos

João instancia o tipo Adulto e, eventualmente, instanciará no futuro o tipo Idoso. Se um indivíduo deve instanciar um determinado tipo em todos os possíveis cenários em que ele possa existir, esse tipo é chamado de Rígido.

Em outras palavras, tipos Rígidos são aqueles que definem características essenciais de suas instâncias. Se todos os indivíduos, que instanciam determinado tipo, podem deixar de instanciá-lo e continuar existindo, esse tipo é chamado Anti-Rígido. Pessoa é um exemplo de tipo Rígido. Tipos Rígidos são estereotipados por «Category», «Collective», «Kind», «Mode», «Quality», «Quantity», «Relator» ou «Subkind». Idoso é um exemplo de tipo Anti-Rígido. Tipos Anti-Rígidos são estereotipados por «Role», «Phase» ou «RoleMixin». Alguns tipos apresentam comportamento rígido em algumas situações e anti-rígidos em outras, sendo neste caso conhecidos como Semi-Rígidos. Tipos Semi-Rígidos são estereotipados por «Mixin», sendo que esses tipos não serão abordados nesse método. A Figura 9 apresenta exemplos de Tipos Rígido, Semi-Rígidos e Anti-Rígidos.

Considere instâncias dos tipos Pessoa e Switch definidos na Figura 5. Se essa pessoa ganhar ou perder peso, ou se esse Switch tiver seu *firmware* atualizado, eles ainda serão instâncias de seus respectivos tipos? Note que um indivíduo João é uma instancia o tipo Pessoa durante toda a sua existência e, ainda que João deixe de existir, o tipo Pessoa continua existindo. Pode-se então adicionar novos tipos ao modelo, de maneira a capturar fenômenos que acontecem na parcela de realidade que se deseje modelar.

Considere a definição de tipo **Switch** apresentada anteriormente, pode-se estender essa definição para um tipo mais complexo adicionando subtipos. Por exemplo, suponha que existam dois subtipos de *switches* no domínio sendo modelado, **Ordinary** e **OpenFlow**. No entanto, todo tipo OntoUML descendente de «Sortal» (vide Figura 4) deve ser um «Substance Sortal» («Kind», «Quantity», ou «Collective») ou ter um como ancestral.

Ao solicitar a verificação do modelo conforme descrito na Subseção 4.1.4 o ambiente editor de ontologias enquanto o diagrama estiver como mostra a parte (a) da Figura 10. Essa restrição desdobra-se do fato de que apenas «Substance Sortal» fornecem um

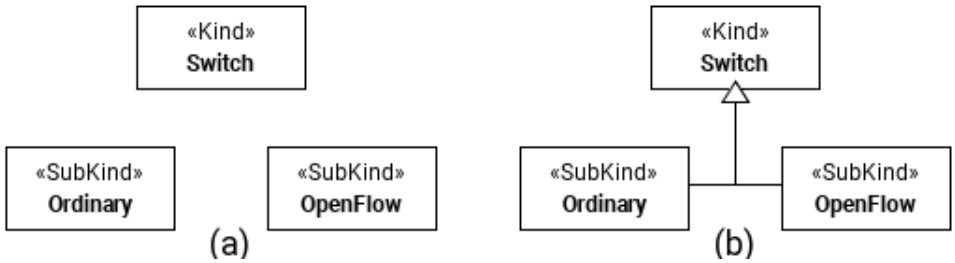

Figura 10 – Tipos Complexos 1

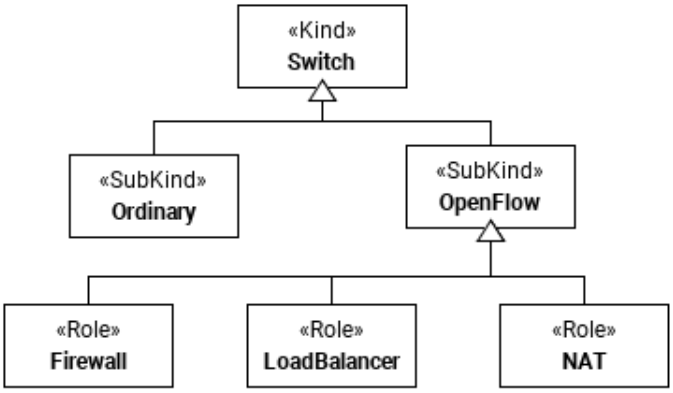

Figura 11 – Tipos Complexos 2

princípio de identidade, sendo os outros «Sortal» existencialmente dependentes destes. Para atender essa restrição, e por conseguinte, eliminar os erros deve-se adicionar dois relacionamentos «Generalization», como mostra a parte (b).

Note que um tipo se liga a seu ancestral através de uma cadeia de relacionamentos «Generalization». Revisitando a noção de comprometimento ontológico a determinada conceitualização a definição de tipo ilustrada na parte (b) da Figura 10 equivale a dizer que o conjunto de fórmulas da ontologia que compõe o modelo contém, entre outras, as seguintes fórmulas:

$$O_1 = \{Ordinary(x) \rightarrow Switch(x)\}$$
$$O_2 = \{OpenFlow(x) \rightarrow Switch(x)\}$$

Pode-se ainda estender essa definição adicionando tipos anti-rígidos («Role» e «Phase»). Por exemplo, enquanto **Ordinary** é apenas um *switch* comum, **OpenFlow** é um *switch* que pode se comportar como um **Firewall**, **LoadBalancer** e/ou **NAT**, desempenhando diversos papéis («Role»). Conforme, exemplificado na Figura 11, ao especificar as generalizações entre os papéis e o subtipo atende-se à restrição de dependência existencial. Porém, papéis devem atender além da existencial, uma outra dependência chamada de relacional.

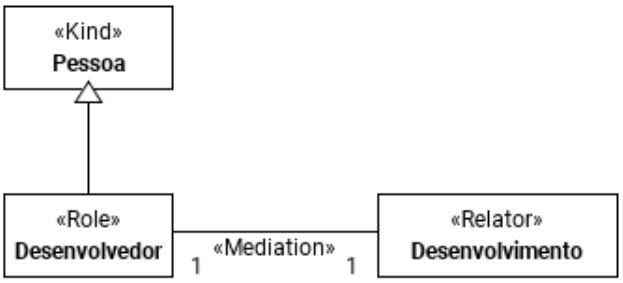

Figura 12 – Dependência Relacional

3.3.5 Dependência

A noção de identidade permite identificar um indivíduo em qualquer circunstância, a de rigidez define quais tipos devem acompanhar o indivíduo em sua existência e quais devem aparecer e sumir de acordo com as mudanças de circunstancia. Como então, distingui-se um conjunto de tipos simples e um tipo complexo.

A distinção a que se faz menção é suportada pelo conceito de Dependência, que é uma noção ontológica importante na definição de tipos complexos. Dependência define como os tipos devem se relacionar. Até aqui explorou-se a definição de tipos classe nos diagramas, mas OntoUML também permite a definição de tipos relacionamento. Existem duas possibilidades de Dependência:

- **Dependência existencial**: é uma dependência observada nos modos intrínsecos («Mode»), tipo que depende de um tipo existencialmente independentes (vide «Substantial» na seção 3.2) por exemplo, um sorriso não pode existir sem um rosto;
- **Dependência relacional**: é uma dependência observada nos modos relacionais («Relator»), tipo que depende de dois ou mais tipos existencialmente independentes, por exemplo, todo papel («Role») e todo modo relacional («Relator») devem estar conectados a pelo menos uma relação de mediação («Mediation»), além disso a soma das cardinalidades nas extremidades opostas às conectadas ao modo relacional deve ser no mínimo dois.

Capítulo 4

Interfacer: Um método para Especificação de Interfaces

Este capítulo descreve o método de especificação de interfaces de comunicação entre componentes de software que implementam partes das arquiteturas SDN e NFV. Enquanto algumas interfaces dessas tecnologias seguem padrões definidos por organizações de padronização, nota-se a necessidade de processos de especificação e implementação de interfaces específicas (CASADO; FOSTER; GUHA, 2014) que se apoiem nas interfaces genéricas afim de acelerar o provisionamento de serviços sobre redes que utilizem essas tecnologias.

Assim, a capacidade de especificar, desenvolver e integrar interfaces específicas para serviços que devem trabalhar em conjunto com interfaces definidas por organizações de padronização torna-se uma melhoria no processo de projeto e desenvolvimento de redes definidas por software (SDN). Esse método segue uma tendência que vem norteando diversas iniciativas como o CIM, YANG e TOSCA, que é o uso de técnicas de modelagem como ferramentas auxiliares no processo de especificação de arquiteturas e implementação de software.

No entanto, esse método se diferencia do que vem sendo praticado nessas iniciativas em dois aspectos. Primeiro, ao associar uma abordagem MDE e padrões MDA com uma técnica de modelagem conceitual orientada a ontologia e modelos especificados usando a linguagem OntoUML. Segundo, pelo foco em interfaces, um aspecto arquitetural que tem recebido atenção devido à sua importância para a evolução das arquiteturas de rede. Ademais, métodos similares tem sido usados em outras áreas como as aplicações web (MILI et al., 2018).

O método é composto por três etapas: a primeira etapa trata da representação do conhecimento e requisitos de interface usando modelos formais; a segunda etapa trata do uso de tecnologias de transformação de modelos em texto para gerar código fonte; e a terceira trata da integração dos códigos fonte gerados automaticamente com os códigos

fonte desenvolvidos tradicionalmente.

Tecnologias que fomentam o uso de software em redes de computadores como SDN e NFV permitem que operadores de rede e provedores de serviço sejam mais inovadores e ágeis ao explorar a virtualização de hardware comum, combinando software de código fonte aberto para atender necessidades de clientes. Esse cenário, entretanto, traz consigo desafios capazes de anular tais benefícios, como o aumento da complexidade de soluções que resulta em problemas de interoperabilidade.

Questões de interoperabilidade podem surgir entre fabricantes, domínios e tecnologias. Considere, por exemplo, que dois operadores de rede distintos tenham interpretações diferentes sobre a arquitetura SDN, de forma que, embora um determinado serviço funcione perfeitamente enquanto utilizado dentro de seu respectivo domínio, ele apresenta problemas ao atravessar a rede do outro operador, sendo que o rastreamento desses problemas beira a impossibilidade devido a sutileza das diferenças.

No que tange a essas tecnologias, organizações de padronização, como IETF, ONF e ETSI, têm trabalhado incessantemente na definição e melhoria de padrões e especificações. É oportuno ressaltar que a interpretação humana desses documentos, frequentemente leva a ambiguidades e erros de entendimento que se propagam por todo o ciclo de vida do software e culmina em produtos ou serviços que enfrentam dificuldade ou são incapazes de operar com outros produtos ou serviços a despeito de serem especificados pelo mesmo padrão.

Diante disso, é apresentado o Interfacer, um método de especificação de interfaces orientado a modelagem que ajuda a reduzir os problemas de interoperabilidade e as divergências de interpretação entre desenvolvedores, provedores de serviço, operadores de rede e fabricantes de equipamentos envolvidos no projeto, desenvolvimento e implantação de arquiteturas e serviços SDN e NFV. O método ajuda a gerenciar a complexidade desses processos pela introdução de artefatos como modelos, procedimentos e ferramentas ao longo do ciclo de vida da solução.

4.1 Representação de Conhecimento

Essa seção trata da primeira etapa do método de especificação de interfaces que consiste na representação do conhecimento. A representação do conhecimento usando linguagens formais e verificáveis como OntoUML é essencial para a promoção da comunicação e do aprendizado dos agentes envolvidos em um processo de desenvolvimento de software. Adicionalmente, essa representação permite que o conhecimento se torne um dos artefatos do processo de desenvolvimento.

Representações de conhecimento são também conhecidas como modelos. A construção desses modelos segue quatro passos que podem ser repetidos indefinidamente até que se atinja a precisão desejada, quais sejam:

1. Caracterização do domínio;
2. Conceitualização;
3. Formalização Ontológica; e
4. Verificação da corretude e consistência do modelo.

Modelos capturam aspectos de um determinado domínio a partir de uma visão abstrata, oferecendo uma representação completa da solução e dos relacionamentos entre os seus componentes, enquanto estabelecem uma linguagem comum a todos os atores envolvidos. Essas linguagens são úteis para especificar atividades de comunicação, soluções de problemas e de aprendizado. Essas atividades, em sua maioria, envolvem interpretação humana, levando a diferentes resultados. A criação de modelos é introduzida no processo de desenvolvimento de sistemas para reduzir variações nos resultados.

O Interfacer usa modelos especificados na linguagem OntoUML, definida a partir de uma técnica de modelagem conceitual orientada a ontologia, que permite, ao mesmo tempo, uma interpretação rápida por humanos treinados, bem como, uma semântica formal para computadores. A modelagem captura a essência do conhecimento da comunidade envolvida e o armazena em modelos em linguagem OntoUML.

4.1.1 Caracterização de Domínio

Caracterização de domínio é uma prática de modelagem que consiste em restringir o escopo do domínio sendo modelado. Restringe-se, portanto, a porção de realidade sendo observada e define-se o subconjunto de fenômenos dessa realidade que são de interesse.

A linguagem OntoUML se apoia em uma ontologia de alta expressividade sendo capaz de representar problemas em áreas de conhecimento distintas como química, física, direito e medicina, entre outras. Para simplificar o processo de modelagem, deve-se deixar claro qual área de conhecimento e qual o problema se deseja representar. Essa caracterização consiste em registrar e informar os agentes envolvidos sobre o problema que se deseja modelar.

Por exemplo, a palavra processo possui significados distintos nas áreas de computação e direito. Na área de direito processo se referre a um conjunto de documentos que subsidiam e registram uma série de decisões. Na área de computação a palavra processo se refere a conjunto de dados que guiam o computador a realizar uma atividade específica. Se necessário, os limites do domínio podem ser ajustados em iterações futuras conforme a evolução do modelo.

Sistemas distribuídos geralmente envolvem diversos atores, como usuários, desenvolvedores, fabricantes, e operadores. A caracterização do domínio deve levar em conta todos esses agentes, suas necessidades e restrições. O ideal é que haja representantes de todas

essas áreas envolvidos no processo de modelagem, ou que quem estiver modelando tenho conhecimento sobre todas elas.

4.1.2 Conceitualização

Uma conceitualização é uma visão abstrata e simplificada do mundo que criamos com algum propósito. Toda base de conhecimento, sistema ou agente relacionado a essa base está comprometido com alguma conceitualização. A despeito da complexa noção psicológica de "conceitualização", ela pode ser explicada usando uma representação matemática bem simples, chamada estrutura relacional extensional. Uma conceitualização ou estrutura relacional extensional é uma tripla $C = (D, W, R)$, onde D é um conjunto de elementos que compõe o universo de Discurso, ou seja, um conjunto de objetos que existem no mundo que queremos conceitualizar, W é o conjunto de possíveis Mundos (*World*), e R é um conjunto de relações conceituais no espaço de domínio $< D, W >$ (GUARINO; OBERLE; STAAB, 2009).

Cada domínio envolve um conjunto dos conceitos de uma área. O análise de um domínio consiste na seleção de um conjunto mínimo de conceitos para a modelagem de um problema específico. Ao minimizar a quantidade de conceitos envolvidos para modelar um problema, as relações entre esses conceitos podem ser determinadas de forma mais simples.

Por exemplo, no domínio da construção civil podemos citar conceitos como fundação, viga, coluna, parede, tijolo, sustentação. Enquanto no domínio da cviação podemos citar conceitos como ar, fuselagem, asa, arrasto, sustentação, leme, bordo. Note que o conceito sustentação aparece no levantamento de ambos os domínios, no entanto, seus significados são completamente distintos. Especificações que se destinem a representá-los devem ser precisas o suficiente para evitar ambiguidade.

O resultado do levantamento de conceitos do domínio é uma lista de conceitos. Esses conceitos são candidatos a tipos que aparecerão nos diagramas que modelam o domínio.

4.1.3 Formalização Ontológica

Formalização ontológica é a atividade de atribuir uma semântica formal às relações conceituais da conceitualização. O conjunto das fórmulas lógicas que atribui semântica de todas as relações conceituais do domínio constitui então, uma ontologia. O conjunto de símbolos das relações conceituais forma o vocabulário de uma linguagem que pode ser usada para descrever possíveis cenários da conceitualização e explicar os fenômenos que ocoorem nele. Assim, essa ontologia garante o comprometimento ontológico entre essa conceitualização e essa linguagem.

A ontologia surge então como uma especificação implícita da conceitualização, restringindo as interpretações dessa linguagem de forma intensional ao aplicar axiomas a

cada predicado. A ontologia, portanto, é o conjunto de fórmulas intensionais, isto é, descrições através dos aspectos mais essenciais associadas a símbolos, de forma que quando um objeto é mencionado, é possível decidir se o objeto pode ou não ser representado por determinado símbolo.

Lidar diretamente com uma ontologia é uma tarefa complexa, que requer treinamento e experiencia. Para facilitar a formação recorre-se ao ambiente de edição de ontologias Menthor (MOREIRA et al., 2016), que permite a especificação de modelos usando a linguagem de modelagem OntoUML. Um modelo OntoUML é composto por diagramas, e um diagrama pode conter vários Tipos. Um Tipo é construído com classes e relacionamentos (ou associações), aplicando as noções ontológicas de identidade (Subseção 3.3.3), rigidez (Subseção 3.3.4) e dependência (Subseção 3.3.5) às relações conceituais. Desse modo, cada modelo contém uma ontologia, que se compromete com suas respectivas conceitualização e linguagem.

4.1.4 Verificação da Especificação do Conhecimento

A ferramenta editora de ontologias Menthor incorpora o meta modelo da OntoUML também especificado no padrão ECore da EMF. Assim, o ambiente conhece as regras sintáticas da linguagem OntoUML. A verificação automática dessas regras é uma característica nativa do ambiente e assegura que os modelos respeitem essas regras. Outro aspecto da verificação, usado para excluir interpretações indesejáveis do modelo, consiste no uso de simulação (BRAGA et al., 2010) criando domínios virtuais chamados de "mundos" e a detecção de anti-padrões.

Anti-padrões são estruturas em OntoUML, conjuntos de classes e relacionamentos interligados de determinada maneira, que reconhecidamente levam a problemas de interpretação.

Os modelos do Interfacer são especificados pela linguagem OntoUML. Ao representar conhecimentos usando uma linguagem apoiada por uma teoria ontológica, restrições podem ser aplicadas para delimitar as interpretações dos modelos. Adicionalmente, a corretude sintática do modelo é verificada automaticamente enquanto o modelo é construído com o editor de modelos Menthor (MOREIRA et al., 2016).

OntoUML foi projetada de tal forma que, por um lado, ela protege os usuários da complexidade de seus princípios ontológicos subjacentes e, por outro, reforça esses princípios provendo mecanismos para a verificação formal automática dos modelos. A sintaxe da linguagem OntoUML tem como base a linguagem Ecore, nome dado ao metamodelo implementado no núcleo da EMF.

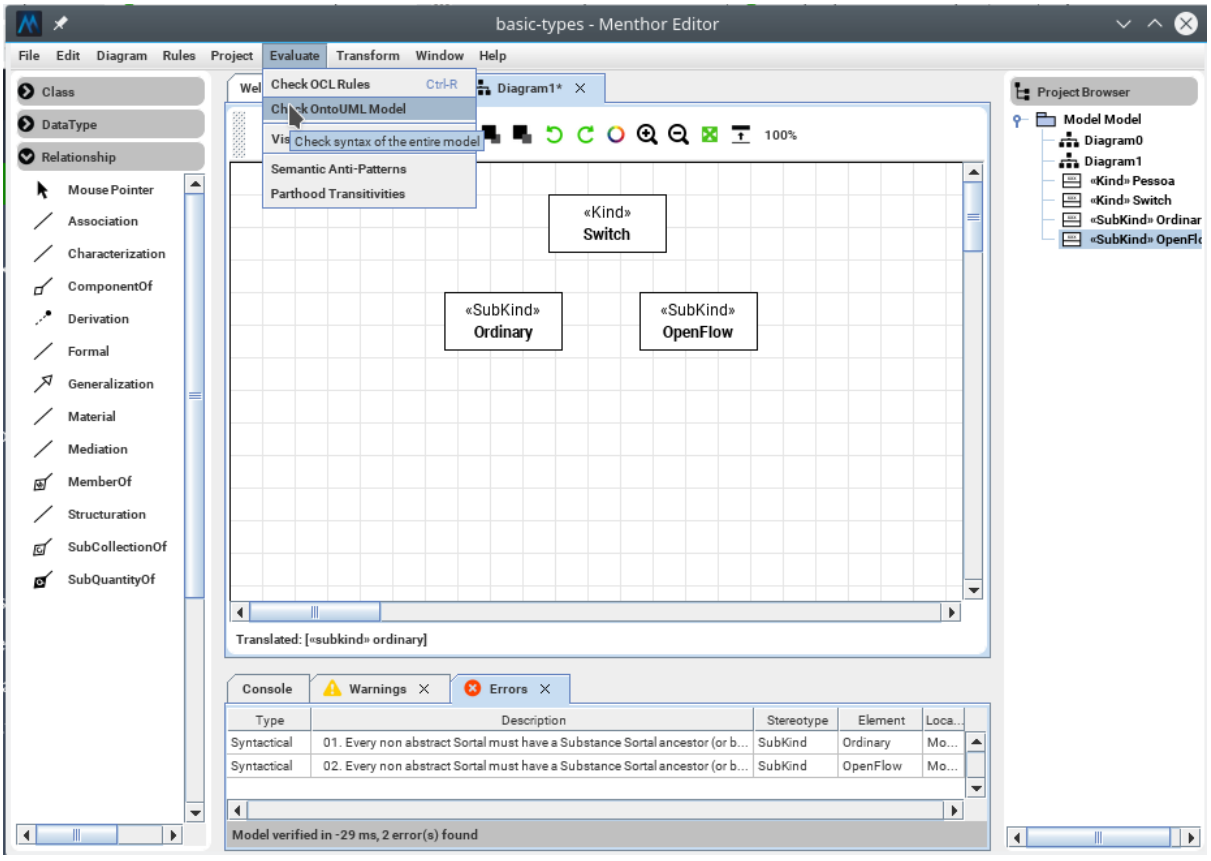

Figura 13 – Verificação de modelos

4.2 Transformação de Modelos em Código Fonte

Essa seção trata da segunda etapa do método de especificação de interfaces, que consiste na transformação do conhecimento representado em modelos em códigos fonte. Essas transformações podem ser usadas para definir um aspecto importante das arquiteturas de soluções baseadas em SDN e NFV: as Interfaces. Interfaces são compostas por um ponto de interação, por abstrações e uma linguagem para manipular essas abstrações.

Para possibilitar a transformação de modelos em trechos de código fonte, esse método emprega tecnologias MDA, como a arquitetura de meta-modelagem *Meta-Object Facility* (MOF) definida pelo *Object Management Group* (OMG), a linguagem *Object Constraint Language* (OCL) e a plataforma EMF.

Transformações são conjuntos de mapeamentos que associam algum elemento do modelo a trechos de código ou atividades do processo produtivo. Transformações recebem um modelo como entrada e entregam parte de um código ou configuração como resultado. Transformações podem ainda ser usadas para promover mudanças no estado de um serviço. O objetivo das transformações é automatizar pequenas atividades do processo produtivo como o desenvolvimento e testes com equipamentos de fabricantes diferentes, configurações diferentes e versões de código diferentes.

O Interfacer usa o *plugin* Acceleo, da interface de desenvolvimento do *Integrated Development Environment* (IDE) Eclipse. Acceleo é uma tecnologia baseada em *templates* de código que inclui ferramentas de autoria para criar geradores de código customizados. Ele permite a produção automática de qualquer tipo de código fonte a partir dos formatos de modelos disponível no EMF. Os *templates* Acceleo capturam o conhecimento armazenado nos modelos e geram trechos de código fonte. Esses *templates* de transformação têm o objetivo de reduzir a dependência da interferência humana no processo de desenvolvimento, reduzindo assim o número de erros de interpretação e digitação.

A definição de transformações é parte essencial desse método. Recomenda-se entretanto, que transformações sejam definidas apenas para os pontos importantes, que sejam definitivamente reconhecidos como padrões ou interface da arquitetura, como estratégia para controlar a complexidade do projeto, uma vez que cada transformação traz consigo uma complexidade que deve sempre estar acompanhada de algum benefício. Pode-se usar o arcabouço EMF para produzir transformações, que produzem estágios intermediários, por exemplo UML, antes de gerar o código fonte apresentado em (PERGL; SALES; RYBOLA, 2013).

4.2.1 Projeto EMF

O projeto EMF é um arcabouço de modelagem e ambiente de geração de códigos para o desenvolvimento de ferramentas e aplicações baseadas em modelos de dados estruturados. A partir de modelos descritos em XML *Metadata Interchange* (XMI), EMF provê ferramentas e suporte em tempo de execução para produzir um conjunto de classes ou trechos de código em diversas linguagens de programação com base no modelo. Além disso, o ambiente disponibiliza mecanismos para visualização e edição dos modelos.

Listing 4.1 – Trecho da definição do Metamodelo OntoUML

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org
   ↪ /2001/XMLSchema-instance"
xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="OntoUML
   ↪ "
nsURI="OntoUML" nsPrefix="OntoUML">
<eClassifiers xsi:type="ecore:EClass" name="Kind" eSuperTypes="
   ↪ #//SubstanceSortal"/>
<eClassifiers xsi:type="ecore:EClass" name="Role" eSuperTypes="
   ↪ #//AntiRigidSortalClass"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="
       ↪ generalization" upperBound="-1"        eType="#//
       ↪ Generalization" volatile="true" transient="true"
```

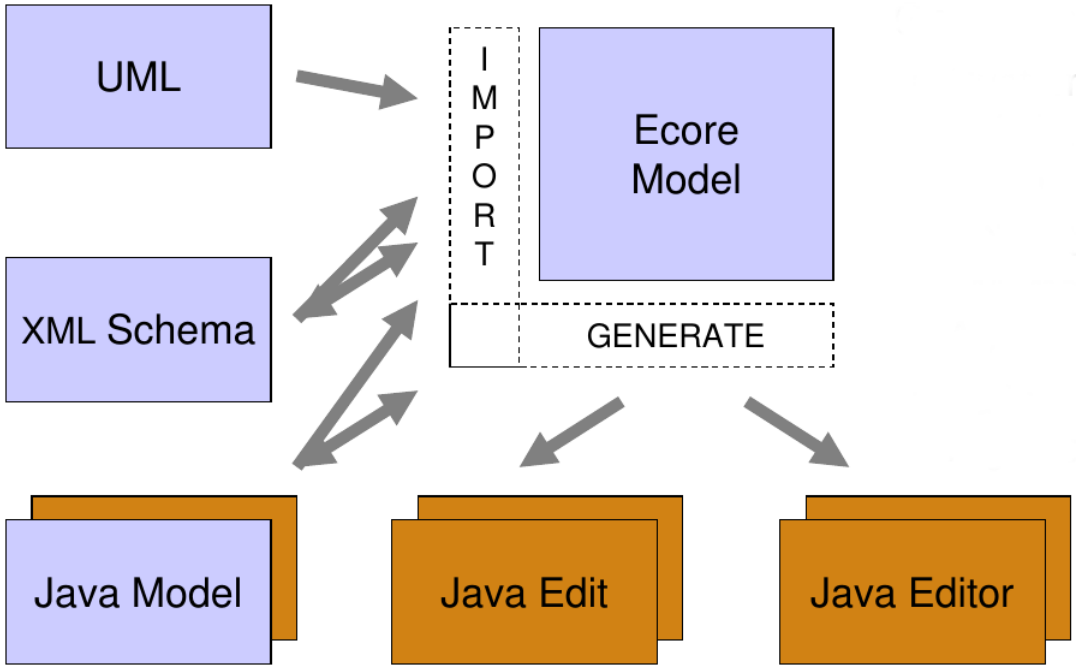

Figura 14 – Importação de Modelo e Geração de Código no EMF (STEINBERG et al., 2009)

```
        ↪ derived="true" containment="true" eOpposite="#//
        ↪ Generalization/specific">
      <eAnnotations source="http://www.eclipse.org/ocl/examples/
        ↪ OCL">
10      <details key="derive" value="Generalization.allInstances
          ↪ ()->select(x␣|␣x.specific␣=␣self)"/>
      </eAnnotations>
    </eStructuralFeatures>
...
```

A especificação da linguagem OntoUML e do meta-modelo OntoUML descritos em Ecore, que é parte do núcleo da EMF, habilita a definição de códigos de transformação usando o *plug-in* Acceleo da interface de desenvolvimento IDE Eclipse da *Eclipse Foundation*. O *plugin* Acceleo permite a especificação em uma linguagem de transformação de modelo para texto, denominada MOF *Model to Text Transformation Language* (MOFM2T).

As Figuras 14 e 15 ilustram o mecanismo de importação de modelos e geração de código e a arquitetura EMF.

4.2.2 Acceleo

Acceleo é uma implementação da linguagem MOFM2T definida pela OMG. Acceleo é implementado por meio da linguagem MTL, composta por dois tipos de estruturas dentro de um módulo: *Templates* e *Queries*. A linguagem permite a criação de expressões usando

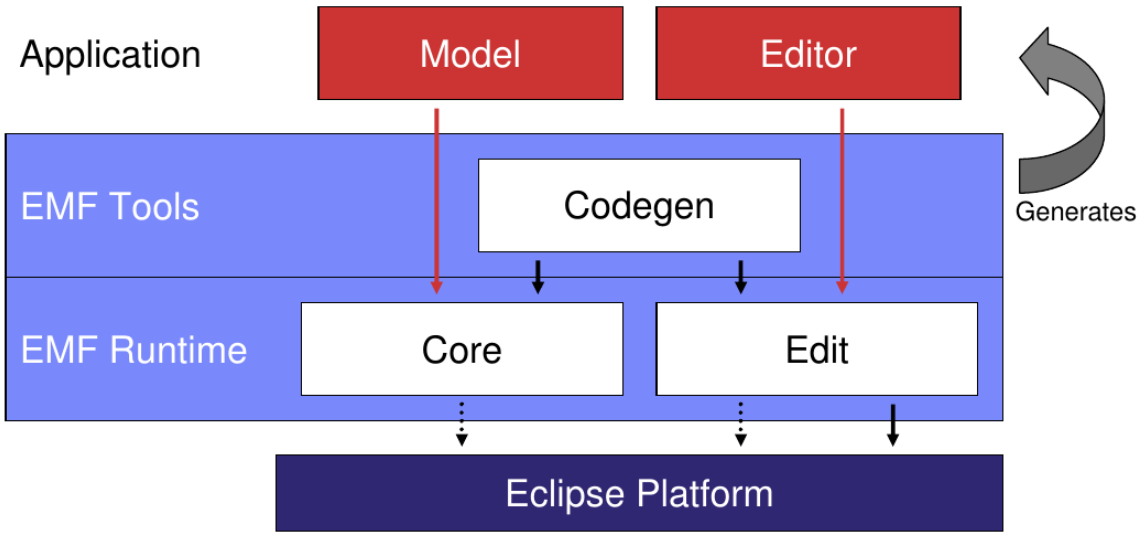

Figura 15 – Arquitetura EMF (STEINBERG et al., 2009)

um subconjunto de OCL para obter informações dos modelos usados como entrada.

Um módulo Acceleo fica armazenado em um arquivo com extensão ".mtl" contendo *templates* para gerar código e/ou *queries* para extrair informações dos modelos manipulados. O início do arquivo deve conter uma declaração de módulo na forma:

$$[module < module\_name > ('metamodel\_URI\_1','metamodel\_URI\_2')]$$

Um módulo pode estender outros módulos, nesses casos os *templates* podem sobrescrever funções definidas nos *templates* importados. Ao importar um módulo, o módulo sendo especificado também ganha acesso a suas consultas.

4.2.3 Linguagem de transformação de modelos

O *plugin* do Acceleo permite escrever *templates* para a geração automática de código fonte. O reconhecimento ou o estabelecimento de padrões ao longo do projeto é essencial para a definição de *templates*. Por exemplo, trechos de código que se repetem, e interfaces de programação orientada a objetos como classes e métodos. Podem ser gerados ainda quaisquer códigos cujas origens dependam de linguagens bem definidas.

Um padrão reconhecido, por estar presente em diversos documentos de especificação de organizações de padronização como IETF, ONF e ETSI, é a definição de interfaces de comunicação entre as partes de uma arquitetura, bem como a definição de uma linguagem de comunicação para manipular as abstrações dentro dessa interface. Propõe-se aqui o uso de transformações para capturar esse padrão e simplificar a implementação de serviços sobre as tecnologias SDN e NFV.

4.3 Integração do Código Fonte a Serviços

Essa seção trata da terceira e última etapa do método de especificação de interfaces. O o *plugin* Acceleo, criado na etapa anterior, deve ser configurado para que cada arquivo de código gerado seja colocado em local específico dentro do repositório de códigos fonte. Dessa forma, os próprios mecanismos de compilação desses projetos se encarregarão em integrar esses códigos no processo de compilação e em reportar eventuais erros.

Os métodos de classes orientadas a objetos, geradas automaticamente, geralmente requerem algum código adicional, para se tornarem totalmente funcionais, tais como ajustes em parâmetros de entrada ou no valor de retorno dos métodos. Além disso, alguns ajustes também serão necessários nos códigos originais que usam o código gerado.

Em caso de compilação bem sucedida, o processo de integração pode incluir a inicialização do serviço e a realização de testes para testar determinados trechos de código ou o serviço como um todo. Essa etapa também prevê a invocação de pré-processadores, compiladores adicionais, configurações e testes ou quaisquer procedimentos necessários a finalização. Simulações, usando a linguagem definida para comunicação dentro da interface, também podem ser realizadas nessa etapa.

As interfaces geradas pelo Interfacer são projetadas para trabalhar em conjunto com as interfaces padrão de arquiteturas SDN e NFV definidas por organizações de padronização. Essas interfaces complementares permitem a definição de serviços sobre os serviços disponíveis em redes definidas por software. As interfaces possuem duas extremidades, (i) uma em contato com os elementos que realizam o controle da rede e (ii) a outra termina em um interpretador que faz a tradução entre a linguagem interna da interface e uma aplicação SDN ou um serviço NFV.

Essa etapa se apoia em aspectos do *Eclipse Modeling Project* (EMP) que reúne padrões e ferramentas para entregar soluções baseadas em modelos com processos para projetar, desenvolver e instalar de fim-a-fim.

4.3.1 Arquitetura SDN

Rede Definida por Software é uma abordagem para gerenciamento de rede que permite configurações dinâmicas de rede programática e eficientemente para adequar a qualidade de serviço e melhorar o monitoramento e o controle de redes. SDN torna a gerencia de rede mais semelhante à computação em nuvem, do que o gerenciamento de rede tradicional. O gerenciamento de rede tradicional permite monitorar a rede, mas o controle é reativo e lento. SDN muda o fato de que a arquitetura estática das redes tradicionais é descentralizada e complexa, enquanto que as redes atuais exigem mais flexibilidade e facilidade na solução de problemas.

SDN centraliza logicamente a inteligência da rede, desacoplando o processo de encaminhamento de pacotes (plano de dados) do processo de roteamento (plano de controle).

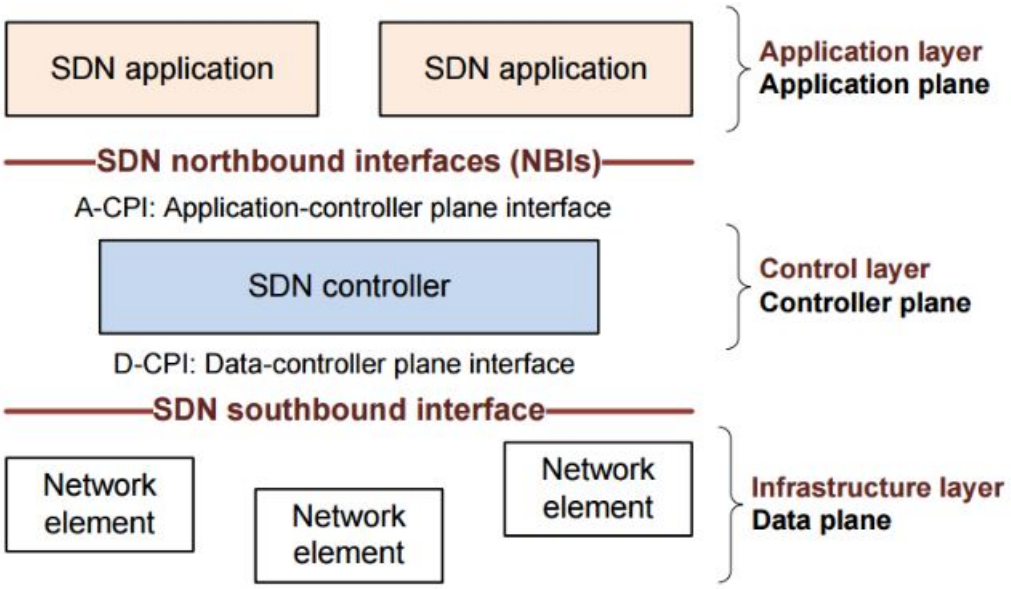

Figura 16 – Arquitetura SDN

O plano de controle pode consistir de um ou mais controladores que são considerados como o cérebro da rede SDN, onde toda a inteligência é incorporada. No entanto, a centralização da inteligência tem suas próprias desvantagens quando se trata de segurança, escalabilidade e elasticidade, que são questões atuais da SDN.

O intuito de SDN é prover interfaces abertas que permitam o desenvolvimento de aplicações de controle que possam controlar a conectividade fornecida por um conjunto de recursos de rede e o consequente fluxo de tráfego de rede através deles. Essas aplicações devem ainda ser capazes de inspecionar e modificar esse tráfego. Essas funções primitivas podem ser abstraídas em serviços de rede que em princípio não são óbvias.

A Figura 16 mostra os componentes básicos da arquitetura SDN composta pelos planos de Dados, Controle e Aplicação. O plano de Dados contém elementos de rede (*Network Element* (NE)), que expõem suas capacidades para o plano de Controle via sua Interface Sul SBI. As aplicações SDN ficam no plano de Aplicação e comunicam seus requisitos para o plano de Controle via Interface Norte NBI. Entre esses dois planos, o plano de Controle traduz os requisitos das aplicações e exerce controle sobre os elementos de rede, enquanto provê informações relevantes sobre o estado da rede para as aplicações.

O controlador SDN pode arbitrar as demandas de aplicações pelo plano de Dados, que competem por recursos de rede limitados, de acordo com políticas estabelecidas.

Desacoplamento entre os planos de Controle e Dados

O princípio de desacoplamento defende a separação dos planos de Controle e Dados. No entanto, fica claro que algum tipo de controle deve ser exercido pelo plano de Controle sobre o plano de Dados. Assim, foi definida uma interface entre controladores e elementos de rede (NE) de forma que controladores possam delegar alguma funcionalidade aos

elementos de rede enquanto mantém informações sobre o estado da rede.

Controle Logicamente Centralizado

Ao contrário de um controle local, o controle centralizado tem uma perspectiva mais ampla dos recursos sob seu controle, podendo tomar decisões melhores sobre como configurá-los. A escalabilidade melhora em função do desacoplamento e da centralização do controle, permitindo uma visão global dos recursos da rede. Lembrando que uma visão global implica uma visão menos detalhada dos recursos da rede.

Exposição de Recursos abstratos e Estados para Aplicações externas

Aplicações podem existir em qualquer nível de abstração ou granularidade, atributos frequentemente descritos em diferentes níveis, dando a ideia de que quanto mais alto o nível, maior o grau de abstração[1]. Como uma interface que expõe recursos e estado pode ser considerada uma interface de controlador, a distinção entre aplicação e controle nem sempre é precisa. A mesma interface funcional pode ser vista sob diferentes visões por diferentes partes interessadas. Assim como os controladores, os aplicativos podem estar relacionados a outros aplicativos como pares ou como clientes e servidores.

O princípio de abstrair recursos de rede e estados de aplicações, através da *Application-Control Plane Interface* (A-CPI), é essencial para que a programabilidade da rede. De posse de informações sobre recursos e seus estados, as aplicações podem especificar requisitos e solicitar alterações em seus serviços de rede por meio do controlador SDN e, portanto, reagir de forma programática aos estados da rede.

4.3.2 Arquitetura NFV

NFV é uma arquitetura de rede, ou até um conceito, que utiliza os fundamentos da tecnologia de TI para virtualizar funções de nós inteiros de redes em servidores, *switches* e armazenamento de alto volume (*Storage*), padrão do setor de telecomunicações, que podem estar localizados em *data centers* ou em *clusters* em empresas. Os nós de rede estão nas instalações do usuário final para criar serviços de comunicação, como ilustrados na Figura 17. NFV envolve a implementação de funções de rede em um software que pode ser executado em uma variedade de hardware de servidor padrão da indústria e que pode ser movido para, ou instanciado em, vários locais da rede, sob demanda, sem a necessidade de instalar novos equipamentos.

O ETSI criou diversos padrões para especificar a virtualização de funções de rede, sendo que aquele apresentado na Figura 17 é um dos mais importantes para o entendimento de NFV. São definidos os aspectos filosóficos e arquiteturais, por meio de camadas, que ajudam a entender como NFV permite o desacoplamento entre o hardware e o software:

[1] Alguns referenciam Níveis como Latitude, introduzindo a ideia de interfaces Norte e Sul

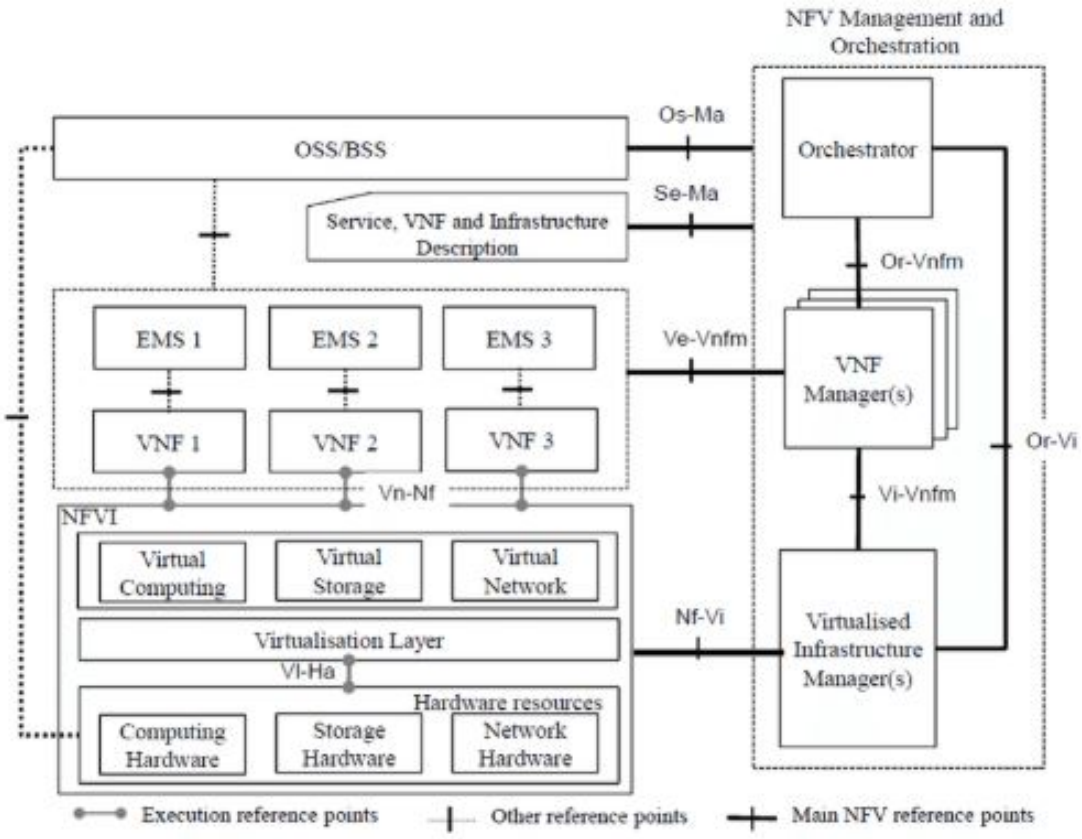

Figura 17 – Arquitetura NFV

1. Camada de Orquestração (*Orchestration Layer*);
2. Camada Função de Virtualização da Rede (*Virtualization Network Function* (VNF) *Manager Layer*);
3. Camada Infraestrutura de NFV (*NFV Infrastructure* (NFVI) *Layer*); e
4. Camada Sistema de Suporte à Operação (*Operation Support System* (OSS) *Layer*).

Como se pode observar na Figura 17, as camadas enumeradas como 1, 2 e 3 compõem um Plano de Gerência e Orquestração (dai o nome *Management and Orchestration* (MANO)), enquanto que a camada OSS/BSS é uma camada de aplicação da arquitetura NFV que interfaceia com MANO por meio da Interface Os-Ma. Observe que a arquitetura NFV especifica um componente para consulta a descrição de Infraestrutura, Serviços e VNF por meio da Interface Se-Ma.

1. Camada de Orquestração (*Orchestration Layer*)

A camada de Orquestração, frequentemente referenciada por seu acrônimo MANO, especifica três componentes: Gerência de Infraestrutura Virtualizada (*Virtualized Infrastructure Manager*), Gerência de VNF e o Orquestrador propriamente dito. Os componentes de gerência podem existir através de uma ou mais instâncias, conforme gerenciem

mais de uma infraestrutura ou VNF. Como contém vários componentes de gerência, por este motivo recebe a referência MANO. Um serviço relevante provido pelo é o serviço de orquestração de rede (SOUSA et al., 2019). A camada MANO interage com ambas as camadas NFVI e VNF, e gerencia todos os recursos da camada de Infraestrutura. MANO cria e remove recursos e gerencia suas alocações nas VNFs.

O *Virtualised Infrastructure Manager* (VIM) compreende as funcionalidades usadas para controlar e gerenciar a interação de uma VNF com recursos de processamento, armazenamento e rede sob sua autoridade, assim como, sua virtualização. O gerenciador de infraestrutura virtualizado executa as seguintes atividades: inventário de recursos, tais como recursos de software, recursos de computação, armazenamento e rede dedicados à infraestrutura NFV; gerenciamento de recursos e alocação de infraestrutura, por exemplo, aumentando *Virtual Machine* (VM)s, melhorando a eficiência energética etc; alocação de VMs em *hypervisors*, recursos de computação, armazenamento e conectividade de rede relevante; análise de causa raiz de problemas de desempenho da perspectiva de infraestrutura do NFV; coleta de informações sobre falhas de infraestrutura; coleta de informações para planejamento, monitoramento e otimização de capacidade, e outras atividades similares.

O gerente de VNFs é responsável pelo gerenciamento do ciclo de vida de VNFs, incluindo instalação, atualização, consulta, escalabilidade (*scale up/down*) e finalização. Uma VNF pode ser gerenciada por um gerente de VNF, numa relação um para um, ou várias VNFs podem ser gerenciadas por um único gerente de VNFs.

O orquestrador é responsável pela orquestração e gerenciamento dos recursos de infraestrutura e de software (NFV), e pela prestação de serviços de rede. Há mais um bloco independente conhecido como *Service*, VNF e Infraestrutura, além dos blocos de construção acima. Isso inclui conjuntos de dados que fornecem informações sobre o modelo de implantação da VNF, gráficos de encaminhamento de VNF, informações relacionadas ao serviço e modelos de informações de infra-estrutura da NFV.

2. Camada Função de Virtualização da Rede

Esta camada é composta por dois componentes VNF e *Element Management System* (EMS). A Função de Rede Virtualizada (VNF) é o componente básico da arquitetura NFV. É considerada uma função de rede virtualizada, por exemplo, quando um Roteador é virtualizado e se torna um VNF *Router*, quando uma estação rádio base é virtualizada (VNF *Base Station* (BS)) e, similarmente, poderia ser um servidor VNF *Dynamic Host Control Protocol* (DHCP) e VNF *Firewall*. Mesmo quando uma subfunção de um elemento de rede é virtualizada, ela é chamada de VNF. Por exemplo, no caso de *Evolved Packet Core* (EPC), várias subfunções como *Mobility Management Entity* (MME), *Gateways* e *Home Subscriber Server* (HSS) podem ser VNFs separados que juntos funcionam como EPC virtualizado.

Uma VNF pode ser instanciada em Máquinas Virtuais (VM) e com isto pode ser movida (migrada) para diferentes locais na rede. Uma VNF pode ser instanciada em várias VMs, onde cada VM hospeda uma única função da VNF. No entanto, toda a VNF também poderia ser instanciada em uma única VM.

O EMS em uma implementação de NFV fornece gerenciamento de rede de componentes VNF e *Physical Network Elements* (PNE). O Gerenciador de VNF notifica o EMS de que precisa para fornecer gerenciamento de elementos para um novo VNF ou PNE. O EMS é responsável pelo gerenciamento de VNFs, que inclui aspectos como falha, configuração, bilhetagem, desempenho e segurança. Geralmente um EMS gerencia várias VNFs, podendo o próprio EMS ser implementado como uma VNF.

3. Camada Infraestrutura da NFV (NFVI)

Uma Infraestrutura de NFV (NFVI) é a totalidade de componentes de hardware e software que criam o ambiente no qual VNFs são implantadas, gerenciadas e executadas. A infraestrutura de NFV pode se estender fisicamente em vários locais, requerendo conectividade física entre esses locais para fazer parte da infraestrutura de NFV. A Infraestrutura de NFV inclui Recursos Virtuais da Camada de Virtualização de Recursos de Hardware. Do ponto de vista da VNF, a camada de virtualização e os recursos de hardware devem ser uma única entidade, fornecendo-lhe o recurso desejado.

Recursos de hardware incluem computadores, armazenamento e rede que fornecem processamento, armazenamento e conectividade a VNFs por meio da camada de virtualização (*hypervisor*). Os recursos de encaminhamento de pacotes e armazenamento são comumente centralizado em um *pool* e organizados para uso compartilhado, enquanto que recursos de rede conectam esses recursos sob demanda. Recursos de rede compreendem funções de encaminhamento de pacotes, como por exemplo, funções desempenhadas por roteadores, *switches* e redes com ou sem fio. Camada de Virtualização também conhecida como *Hypervisor*, abstrai os recursos de hardware e separa o software da VNF do hardware subjacente para garantir um ciclo de vida independente de hardware. Esta camada é responsável principalmente pelo seguinte: abstração e particionamento lógico de recursos físicos.

Permitindo que o software instancie a VNF, para usar a infraestrutura de virtualização subjacente que fornece recursos virtualizados, permite que ela possa ser executada independentemente da localização. A camada de virtualização no meio da arquitetura garante que VNFs sejam desacopladas de recursos de hardware, permitindo que o software seja implantado em diferentes recursos físicos.

4. Camada Sistema de Suporte à Operação (OSS)

OSS e *Bussiness Support System* (BSS) são dois importantes sistemas para tratar com os aspectos operacionais e do negócio de operadoras. OSS lida com aspectos tais

como gerenciamento de rede, gerenciamento de falhas, gerenciamento de configuração e gerenciamento de serviços. BSS lida com aspectos ligados a gerenciamento de clientes, gerenciamento de produtos, gerenciamento de pedidos etc.

Na arquitetura NFV, o desacoplamento do BSS e OSS permite que uma operadora os integre ao Gerenciamento e Orquestração da NFV usando interfaces padrão. É interessante notar que na arquitetura NVF, a camada OSS/BSS é uma camada de aplicação, que traduz as políticas de negócios e operação da empresa, fazendo uso dos serviços da camada VNF e interagindo com MANO através da interface Os-Ma.

4.4 Conclusão

O presente capítulo apresentou o método especificado para a criação de interfaces que são disponibilizadas pelo Interfacer. No futuro, há uma visão de que o Interfacer possa ser disponibilizado como um repositório de interfaces, nos moldes do que acontece com as lojas de aplicativos para *smartphones*.

A definição de um método preconiza a especificação dos passos que devem ser seguidos para se atingir um objetivo, neste caso, a criação de uma interface. Para cada um dos passos foram especificados os conceitos, ferramentas e procedimentos para seus cumprimentos. No caso do Interfacer, podemos resumir o método em três grandes passos:

1. **Representação de Conhecimento**

   Esta primeira etapa do método de especificação de interfaces (seção 4.1) consiste na representação formal do conhecimento usando linguagem OntoUML. A representação formal e verificável é essencial para a promoção da comunicação e do aprendizado dos agentes envolvidos em um processo de desenvolvimento de software. Adicionalmente, essa representação permite que o conhecimento se torne um dos artefatos do processo de desenvolvimento.

2. **Transformação do Modelo em Código Fonte**

   Esta segunda etapa do Interfacer (seção 4.2) consiste na transformação do conhecimento representado em modelos, levantados no passo anterior, em códigos fonte. Essas transformações podem ser usadas para definir um aspecto importante das arquiteturas de soluções baseadas em SDN e NFV: as Interfaces. Para possibilitar a transformação de modelos em trechos de código fonte, esse método emprega tecnologias MDA, como a arquitetura de meta-modelagem MOF definida pelo OMG, a linguagem OCL e a plataforma EMF.

3. **Integração do Código fonte a Serviços**

   Esta terceira e última etapa do Interfacer (seção 4.3) consiste em utilizar o projeto do *plugin* Acceleo, criado na etapa anterior, para que cada arquivo de código gerado

seja colocado em local específico dentro do repositório de códigos fonte. Dessa forma, os próprios mecanismos de compilação desses projetos se encarregarão em integrar esses códigos no processo de compilação e em reportar eventuais erros. O processo de integração inclui a inicialização do serviço e a realização de testes para testar determinados trechos de código ou o serviço como um todo. Essa etapa também prevê a invocação de pré-processadores, compiladores adicionais, configurações e testes ou quaisquer procedimentos necessários a finalização. Simulações, usando a linguagem definida para comunicação dentro da interface, também podem ser realizadas nessa etapa.

A integração do código fonte a serviços é feita mediante o mapeamento de serviços especificados por organizações de padronização para as interfaces de arquiteturas SDN, tais como as interfaces SBI, NBI, *East Bound Interface* (EBI) e *West Bound Interface* (WBI), e as interfaces de arquiteturas NFV, tais como as interfaces Os-Ma, Se-Ma, Ve-Vnfm, Nf-Vi.

Capítulo 5

Interfacer aplicado à Arquitetura ETArch

A facilidade introduzida pelo Interfacer, de oferecer interfaces especificadas para domínios de aplicação, criadas para uma finalidade bem definida, é um anseio antigo e que se torna mais agudo num cenário de Internet do Futuro. Como exposto no capítulo 2, existem várias iniciativas com vistas a redes para Internet do Futuro, sendo que aquelas baseadas na filosofia SDN apresentam as condições adequadas para a programabilidade da infraestrutura de rede.

Para a verificação do Interfacer, foi escolhida a arquitetura ETArch, projetada no seio do programa *Mondial Entities Horizontally Addressed by Requirements* (MEHAR), que está operacional desde 2012. Desde então vem sendo melhorada com vistas a oferecer serviços mais ricos e, portanto, oferece um ambiente propício para a verificação do Interfacer. A seção 5.1 descreve o método utilizado para avaliar a aplicabilidade do Interfacer. A seção 5.2 especifica os meios para representação dos conceitos da arquitetura utilizada. A seção 5.3 descreve o passo para a transformação de conhecimentos em código fonte. A seção 5.4 apresenta o modo como o código fonte é integrado e fica disponibilizado por meio de serviços.

5.1 Método para a Avaliação

O método de avaliação consiste na comparação qualitativa, primeiro das especificações que representam o MTE, em aspectos como expressividade, extensibilidade e usabilidade. Segundo, dos processos de desenvolvimento de software usados para implementar a ETArch, em aspectos como aderência à arquitetura SDN, e adaptação a novas tecnologias.

O presente método deve ser capaz de substituir e melhorar todo o ciclo de vida de uma solução de rede baseada em SDN, que compreende a especificação em alto nível de abstração apresentada em (PEREIRA, 2012), transformada na especificação a nível de

rede de computadores apresentada em (SILVA, 2013).

Aplica-se os principais critérios (funcionalidade, usabilidade, eficiência, manutenibilidade e portabilidade) do padrão ISO/IEC 25010:2011, que trata dos requisitos de qualidade e avaliação e sistemas e software, afim de determinar se o presente método obteve melhoria em relação ao método anterior.

5.2 Representação dos Conceitos da Arquitetura

Essa seção descreve como a etapa de representação de conhecimento provida pelo Interfacer foi aplicada ao MTE, que originalmente guiou a primeira implementação da ETArch. Primeiro é realizada a caracterização do domínio, seguida pela conceitualização do MTE, onde os conceitos fundamentais do modelo são revisitados. De posse desses conceitos, passa-se então à formalização ontológica, para finalizar com a verificação, afim de garantir a corretude do modelo, resultando em uma especificação mais precisa do MTE.

5.2.1 Caracterização do Domínio

Embora diversos aspectos até aqui apresentados, como o modelo MTE, a arquitetura ETArch e a arquitetura SDN, contribuam para a caracterização do domínio desse experimento, define-se inicialmente, o MTE como a porção de realidade sendo observada e os conceitos de entidade, título e workspace como subconjunto de fenômenos de interesse. Considera-se o MTE como assunto prioritário, visto que ele abriga a filosofia, conceitos, definições e informações da arquitetura que se deseja especificar, embora em alto nível de abstração, uma vez que é um modelo.

O MTE é, portanto, o ponto de partida para a obtenção de uma especificação mais precisa dessa arquitetura, de onde será posteriormente extraído o conhecimento para complementar as futuras interfaces. Deste modo, se restringe inicialmente ao MTE o domínio de aplicação do Interfacer, sendo que outros aspectos poderão ser inclusos à medida que forem necessários.

5.2.2 Conceitualização

Conforme definido na subseção 4.1.2, uma conceitualização é uma visão abstrata e simplificada do mundo que criamos com algum propósito. Toda base de conhecimento, sistema ou agente relacionado a essa base está comprometido com alguma conceitualização. Recupera-se aqui as definições apresentadas na seção 2.2 para constituir a conceitualização do MTE.

O Modelo de Título de **Entidade** enriquece o conceito de entidade definida no modelo de referência OSI, que define entidade todo **ser** que tem capacidade de se **comunicar** num ambiente distribuído. De fato, a semântica de Entidade se origina no Grego, para

se referir a um 'ser que existe'. No MTE, **Entidades** têm necessidades para suportar suas comunicações, podem herdar propriedades e manter relações com outras entidades. Como seres comunicantes, no Modelo de Título, **Entidades**, além de necessidades, têm capacidades que podem ser utilizadas em contextos de comunicação.

O conceito de **Título** foi definido no Modelo de **Título** de Entidade para identificar elementos de comunicações que precisam ser identificáveis durante seus ciclos de vida no ambiente distribuído. **Títulos** são, portanto, identificadores abstratos, unívocos, não ambíguos e independentes da localização do elemento que identificam. Ao ser abstratos, e portanto de um alto nível de representação, **Títulos** independem da topologia e da natureza da infraestrutura de rede que suportam os elementos identificáveis.

O Modelo de Título de Entidade define o conceito de ***Workspace*** para representar o domínio de comunicação capaz de suportar as necessidades de comunicação de Entidades participantes. ***Workspaces*** são capazes de suportar as relações de entidades (*Unicast*, *Multicast*, *Broadcast*) de forma natural. De acordo com o Modelo de Título de Entidade, ***Workspace*** pode ser definido como um barramento lógico que independe de topologias e natureza da infra estrutura de redes interconectadas.

5.2.3 Formalização Ontológica

Uma vez levantados os principais conceitos relativos a arquitetura, fez-se uma análise das propriedades ontológicas desses conceitos chegando a três diagramas distintos, (i) um que modela o conceito de entidade, (ii) outro que modela o conceito de dispositivo, e um terceiro que modela o relacionamento de (i) e (ii) numa relação de hospedagem.

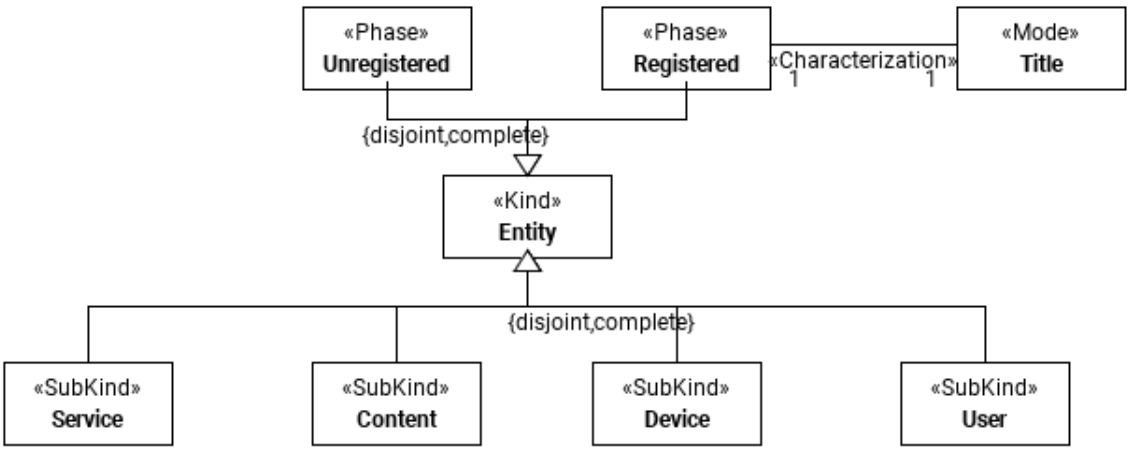

Figura 18 – Diagrama de entidade (*Entity*)

Uma das principais relações conceituais do MTE é a Entidade. O propósito de uma entidade é se comunicar com outras entidades através de um serviço de comunicação. A

despeito disso, uma entidade continua existindo, mesmo quando não está se comunicando, mesmo ainda, quando está desconectada ou fora do alcance do serviço de comunicação.

Nesse sentido, uma Entidade é existencialmente independente, e deve além disso, fornecer um princípio de Identidade, justificando desse modo a classe base estereotipada como «Kind» e nomeada de *Entity*. Adiciona-se um subtipo («Subkind») para cada abordagem de Internet do Futuro, conforme previsto na subseção 2.2.2 e representado na Figura 18. Como as classes (*Content, Device, Service* e *User*) não são «SubstanceSortal», então devem ter esta classe como ancestral, e como são rígidas («Rigid Sortal»), e não fornecem um princípio de identidade, devem herdar aquele fornecido por *Entity*. Essas classes devem, portanto, estar associadas à classe *Entity* por relacionamentos de generalização («Generalization»).

Para ingressar no, ou deixar o, serviço de comunicação, uma entidade deve negociar com a rede, de forma a obter, ou dispensar, as autorizações e recursos necessários ao provisionamento do serviço. Essa negociação evidencia que uma entidade pode sofrer alterações durante a sua existência. Essas alterações geralmente são representadas por classes anti-rígidas («Anti-Rigid Sortal» como «Role» ou «Phase»). Essas alterações são modeladas ao longo de diversas etapas.

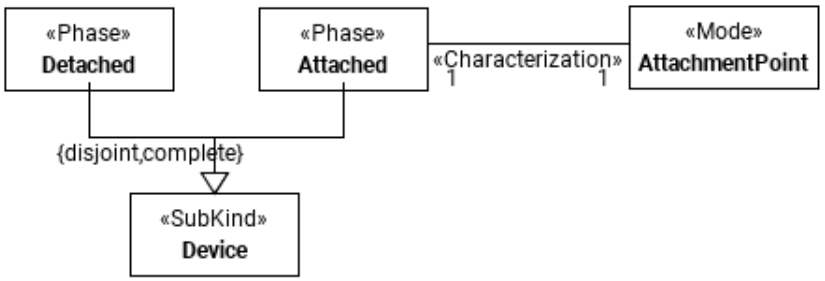

Figura 19 – Diagrama de dispositivo (*Device*)

A primeira etapa ataca as alterações físicas, uma vez que dispositivos, tais como computadores, celulares, carros e eletrodomésticos, estão fisicamente ligados à rede, ou seja, todos eles implementam algum tipo de camada física que lhes concede acesso à rede. Modela-se essa questão no diagrama de dispositivo, ao adicionar as fases («Phase») *Attached* e *Detached*, conforme ilustrado na Figura 19, para indicar que um dispositivo pode estar conectado ou não à rede.

Do ponto de vista lógico, uma entidade pode assumir estados diferentes, dependendo da disponibilidade de recursos do serviço de comunicação. As fases («Phase») *Registered* e *Unregistered* são adicionadas ao tipo *Entity*, modelando esse fenômeno. Além disso, um Título é percebido como uma característica existencialmente dependente de uma entidade, representado, portanto, como um modo («Mode») e associado por um relacionamento

de caracterização («Characterization») à fase *Registered*. Em outras palavras, somente entidades registradas no DTS possuem Título(s).

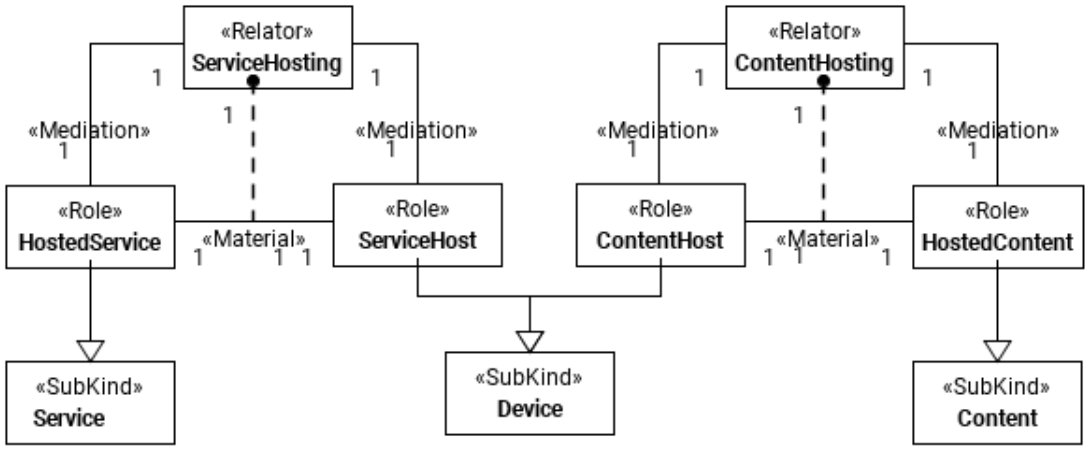

Figura 20 – Diagrama de hospedagem (*Hosting*)

Subtipos diferentes de entidade (*Content*, *Device* e *Service*) podem desenvolver relacionamentos especiais entre si. Por exemplo, um dispositivo pode conter diversos conteúdos e serviços. No entanto, um relacionamento entre um dispositivo e um conteúdo (um celular e uma foto) não pode existir senão em função da existência de ambos. Isso significa que o tipo que representa esse relacionamento deve não apenas ser existencialmente, mas relacionalmente dependente. Modela-se essa situação através de modos relacionais («Relator»).

Introduz-se então um novo diagrama para representar a ideia de hospedagem (*Hosting*) e são acrescentados dois modos relacionais («Relator»), sendo 'hospedagem de conteúdo' *ContentHosting* e 'hospedagem de serviços' *ServiceHosting*. Adiciona-se ainda os papéis («Role») de hospedeiro (*ContentHost* e *ServiceHost*) e de hóspede (*HostedContent* e *HostedService*), que são relacionalmente dependentes dos modos relacionais. Assim, um 'dispositivo' só se torna hospedeiro e 'conteúdos e serviços' só se tornam hóspedes durante a existência de uma hospedagem, conforme é apresentado na Figura 20.

Workspace é um relacionamento entre pelo menos duas aplicações (entidades) que estão registradas e hospedadas em dispositivos também registrados e anexados à rede. O registro de aplicações e dispositivos exige que um usuário registrado se responsabilize por eles. O estabelecimento do *Workspace* implementa conectividade com difusão seletiva e mobilidade naturais, uma vez que todos os dispositivos na rede são rastreáveis, permitindo que seus participantes se comuniquem.

Workspace, como um «Relator» é um estereótipo que caracteriza um tipo provedor de identidade e rígido, mas cuja instância depende existencialmente das instâncias dos tipos relacionados com ele. Assim um *Workspace* só pode existir entre pelo menos duas aplicações hospedadas. Uma aplicação hospedada depende de um hospedeiro com recur-

sos disponíveis para realizar a hospedagem. Hospedeiro é um papel que um dispositivo desempenha. Um dispositivo se anexa na rede necessariamente em algum ponto, dispondo portanto, de um ponto de anexação.

5.2.4 Verificação da Especificação em OntoUML

A ferramenta editora de ontologias Menthor (MOREIRA et al., 2016) incorpora o meta modelo da OntoUML, também especificado no padrão ECore da EMF. Assim, o ambiente conhece as regras sintáticas da linguagem OntoUML. A verificação automática dessas regras é uma característica nativa do ambiente e assegura que os modelos respeitem essas regras. Conforme já mencionado na subseção 4.1.4, a verificação de regras sintáticas acontece automaticamente à medida que se constrói o modelo.

A Figura 21 mostra a simulação de um cenário instanciado de acordo com o diagrama de hospedagem (*Hosting*) com auxílio da linguagem *Alloy* (BRAGA et al., 2010) no ambiente de edição de ontologias Menthor. Essa simulação ajuda a identificar interpretações indesejadas do modelo, como, por exemplo, cenários que não condizem com o que acontece, ou deve acontecer, na realidade. Essas interpretações indesejadas devem ser eliminadas, aumentando a precisão do modelo através de sucessivas iterações da etapa de representação de conhecimento.

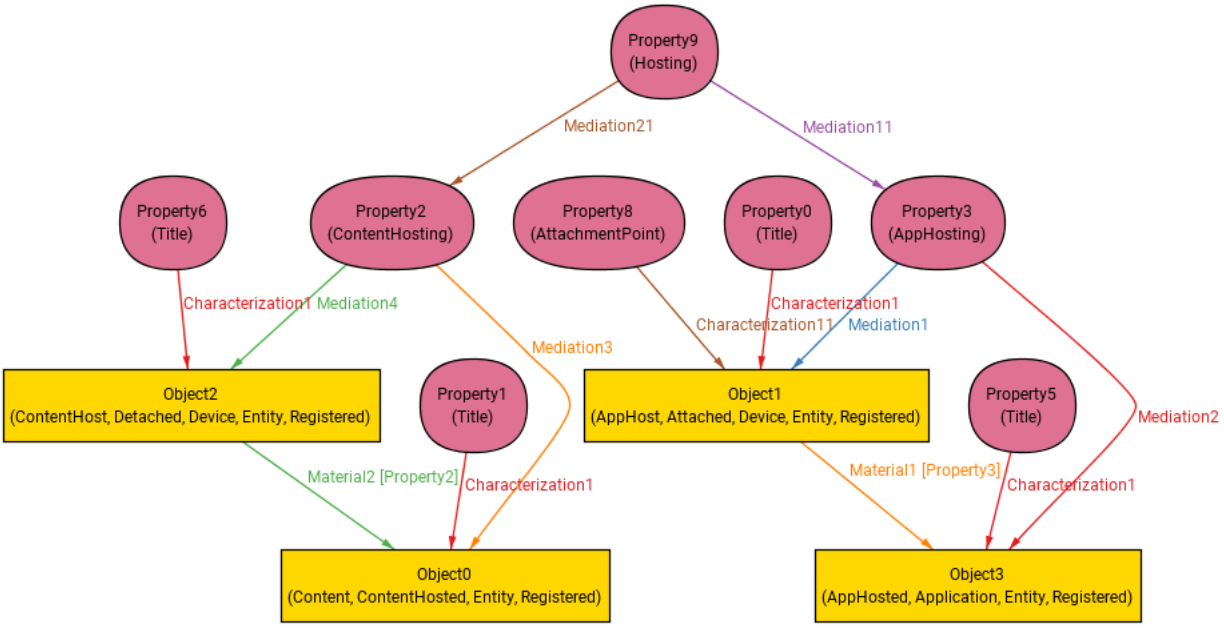

Figura 21 – Simulação do diagrama de hospedagem (*Hosting*)

Note que o cenário simulado instancia objetos cujos predicados são as classes que definem os tipos construídos nos diagramas que compõem a especificação (modelo) do MTE em OntoUML. A inscrição (*ContentHost, Detached, Device, Entity, Registered*) do objeto dois (*Object2*) indica que ele é uma entidade do subtipo dispositivo, que se registrou,

que desempenha o papel de hospedeiro, e no momento, o serviço de comunicação considera que está desconectado da rede.

A Figura 21 mostra ainda que, por estar registrado, o objeto dois é caracterizado pelo Título (*Property6*). Enquanto dispositivos estão ou não conectados à rede, conteúdos e serviços só têm acesso ao serviço de comunicação durante sua hospedagem em algum dispositivo. Nesse cenário, o objeto *Object0* é uma entidade do subtipo *Conteúdo*, que está hospedada no objeto *Object2*, e neste momento está registrada. A relação de hospedagem de conteúdo (*Material2*) deriva da propriedade de hospedagem de conteúdo (*Property2*), que media o hóspede e o hospedeiro.

5.3 Transformação do Modelo em Código Fonte

Essa seção descreve como a etapa de transformação de modelo em código fonte provida pelo Interfacer foi aplicada ao modelo especificado na seção 5.2. Códigos intermediários foram definidos em MTL para extrair conhecimento da nova especificação e gerar códigos fonte para a nova interface do DTS, para o módulo do Floodlight e para a interligação desses componentes.

5.3.1 Especificação da interface do DTS

Na implementação original da ETArch, os protocolos do plano de Controle são responsáveis pelo ciclo de vida de Entidades e *Workspaces* e oferecem serviços tais como: registro de uma Entidade no DTS; criação de um *Workspace*; entrada (*attach*) e saída (*detach*) de entidades em workspace etc. O *Entity Title Control Protocol* (ETCP) oferece serviços que envolvem interações entre Entidades e o DTS (Plano de Controle), enquanto o DTS *Control Protocol* (DTSCP) é responsável por interações entre DTSAs ou *Master DTSA* (MDTSA)s, que são essencialmente o Plano de Controle, com o intuito de prover alta disponibilidade e escalabilidade. O protocolo ETCP é disponibilizado pelo Plano de Controle para disciplinar a relação entre Entidades e DTSAs, ou seja, o ETCP é a interface disponível às entidades para suas solicitações ao DTS. O protocolo DTSCP é responsável pelas relações de controle internas ao Plano de Controle (DTS), entre DTSAs e MDTSAs.

Ao invés de focar em questões de desempenho como, alta disponibilidade e escalabilidade, decidiu-se focar em aumentar a qualidade do processo de desenvolvimento, priorizando aspectos como aderência entre modelo e código fonte, expressividade e reusabilidade. Assim, a arquitetura de implementação foi refatorada, de forma que apenas umDTSA provê o serviço do DTS e o protocolo DTSCP não é mais necessário.

Foca-se, portanto, em construir uma transformação que aproveite a especificação mais expressiva do conceito de entidade para gerar automaticamente um código que reconheça

sentenças da gramática que define a linguagem de comunicação entre entidades comunicantes e o DTS. Desse modo, o protocolo ETCP foi substituído por um interpretador. Essa transformação foi construída sobre um *parser* na linguagem de programação *Ocaml*, gerado pelo gerador de *parser* Menhir.

Essa primeira transformação gera o código fonte responsável pela linguagem de comunicação entre o controlador OpenFlow Floodlight e o DTSA. As classes que representam predicados temporários estereotipados com («Phase»), tais como *Registered, Unregistered, Attached* e *detached* são transformadas em símbolos da linguagem que representam ações sobre entidades. Nesse sentido, uma entidade pode ser registrada em um domínio de Título ou ainda, uma entidade pode ser anexada ou desanexada de um *Workspace*.

Outro aspecto importante dessa linguagem de interface é a definição do formato do Título baseado no formato do endereço físico das interfaces de rede *Ethernet*, composto por seis pares de algarismos hexadecimais. Dessa maneira toda entidade registrada deve ser identificada por um Título unívoco, conforme especificado pelo MTE. Decidiu-se por utilizar este formato para título, pois, deste modo, é possível utilizar as placas de redes legadas para as verificações, lembrando placas de rede *Ethernet* são o padrão de *facto* encontradas em estações de trabalho.

5.3.2 Módulo do controlador Floodlight

Outro padrão que foi incorporado em uma segunda transformação foi o módulo do controlador OpenFlow Floodlight. O Floodlight é um controlador SDN de alto desempenho desenvolvido em Java. Sua arquitetura prevê uma composição modular, de modo que o mecanismo, para gerenciamento de seus módulos e as interfaces dos módulos com o controlador, já se encontra bem maduro. Como o Floodlight é implementado em Java e a IDE Eclipse provê a geração automática de classes Java baseadas em interface (interface da linguagem de programação Java), a incorporação do código gerado em um *template* Acceleo, uma cria classe Java que extende as interfaces e métodos Java apropriados para implementar um módulo do Floodlight.

5.3.3 Comunicação entre Floodlight e DTSA

Pelo fato do Floodlight ser implementado em linguagem Java e o interpretador da linguagem da interface ser implementado em linguagem Ocaml, optou-se por usar o Apache Trift para interligar os dois componentes. Apache Thrift é um *framework* para desenvolvimento de serviços onde as pontas usam linguagens diferentes. Usando a especificação do Apache Thrift, uma transformação Acceleo define as estruturas de dados e os estados que permitem que os componentes façam chamadas remotas entre si independentemente da linguagem de programação que os implementa.

```
namespace java net.floodlightcontroller.etarch.xpsa.thrift
```

```
namespace ocaml xpsa

service Interpreter {

string interpret(1:string code)

}
```

O fragmento de código acima é uma especificação entre o módulo do Floodlight e o interpretador da linguagem da interface. Esse interpretador é o componente do novo DTSA, que doravante é a interface entre Entidades (usuárias do Plano de Dados) e o DTS (Plano de Controle).

5.4 Integração do Código Fonte em Serviços

Essa seção descreve como a etapa de integração de códigos fonte em serviços, provida pelo Interfacer, aproveita a infra-estrutura do ambiente de desenvolvimento orientado a modelos, para integrar as interfaces geradas a partir do modelo com as interfaces padrão da arquitetura SDN.

5.4.1 Integração entre DTSA e rede SDN via NBI

A implementação original da ETArch utilizava a infraestrutura do *JAIN SLEE*, uma plataforma implementada em Java, para o desenvolvimento de aplicações de telecomunicações de alto desempenho. Assim o acesso a recursos do controlador da rede, que compunha o DTSA, era feito via *Application Programming Interface* (API) Java. A nova implementação busca melhor aderência com a especificação de arquiteturas SDN e acessa os recursos do controlador via NBI, conforme pode ser visto no *script* a seguir.

Listing 5.1 – Integração DTSA e Floodlight via NBI

```
#!/usr/bin/python

import os
import json

silent = "-s"
nbi_address = "localhost:8080"
devices_uri = "wm/device/"
routing_uri = "wm/routing/path/"
switches_uri = "wm/core/controller/switches/json"
sep_uri = "wm/staticentrypusher/json"
flows_uri = "wm/core/switch/all/flow/json"
```

```
15    def devices():
        command = "curl␣%s␣␣http://%s/%s" % (silent, nbi_address,
           ↪ devices_uri)
        data = json.loads(os.popen(command).read())
      # print type(data)
      # print data
20      result = []
        for device in data:
          #for entity, vlan, ap, mac, ip, lastseen in device:
      #     print device['mac']
            for ap in device['attachmentPoint']:
      #       print ap['switchDPID'], ap['port']
              result.append((device['mac'][0], ap['switchDPID'], ap['
                 ↪ port']))
        return result


30    def routing(sw1, p1, sw2, p2):
        command = "curl␣%s␣http://%s/%s%s/%s/%s/%s/json" % (silent,
           ↪ nbi_address, routing_uri, sw1, p1, sw2, p2)
        print command
        data = json.loads(os.popen(command).read())
        print data
35


      print "Devices:\n"
      command = "curl␣%s␣␣http://%s/%s" % (silent, nbi_address,
         ↪ devices_uri)
      result = os.popen(command).read()
40    print command+"\n"
      print result+"\n"


      print "Switches:\n"
45    command = "curl␣%s␣␣http://%s/%s" % (silent, nbi_address,
         ↪ switches_uri)
      result = os.popen(command).read()
      print command+"\n"
      print result+"\n"
```

```
50
    print "Flows:\n"
    command = "curl␣%s␣␣http://%s/%s" % (silent, nbi_address,
       ↪ flows_uri)
    result = os.popen(command).read()
    print command+"\n"
55  #print result+"\n"

    devlist = devices()
    routing(devlist[0][1], devlist[0][2], devlist[2][1], devlist
       ↪ [2][2])
```

Note que a invocação de interfaces pode ser facilmente mantida, bastando adequar as linhas do *script* que contêm o comando **curl**.

5.4.2 Integração do servidor *Apache Thrift* e DTSA

Esse código demonstra como o servidor do *Apache Thrift* foi integrado ao interpretador do DTSA. Esse interpretador trata a linguagem de comunicação entre entidades e o DTS, que é parte da interface definida pelo Interfacer para intermediar o serviço de Título e Entidades, O *Apache Thrift* gera códigos em diversas linguagens de programação baseados em uma definição de interface de serviço. Requisições e Entidades e respostas do DTS são trocadas através dessa interface de serviço.

Isso permite que uma solução seja implementada usando linguagens de programação diferentes e mais apropriadas a cada tarefa. O interpretador, por exemplo, foi desenvolvido na linguagem de programação OCaml. Assim, foi gerada uma biblioteca contendo o servidor *Apache Thrift* e a interface de serviços implementada em OCaml.

Listing 5.2 – Integração lado DTSA

```
open Ast
open Printf
open Thrift
open Xpsa_types
5
(* Parse a string into an ast *)
let parse s =
  let lexbuf = Lexing.from_string s in
  let ast = Parser.grammar Lexer.read lexbuf in
10 ast


let is_result : exp -> bool = function
```

```
    | String _ -> true
    | Title _ -> true
  (*  | Relation _ -> true *)
    | _ -> false


let step : exp -> exp = function
    | Title _               -> failwith "Does not step"
  (*  | Relation _            -> failwith "Does not step" *)
    | _       -> failwith "Error"


let rec eval : exp -> exp = fun e ->
  if is_result e then e
  else eval (step e)


let extract_value = function
    | String s -> print_string (String.concat "" (s :: "\n" :: []))
    | Title t  -> print_string (String.concat "" (t :: "\n" :: []))
    | _ -> failwith "Not a value"


(* Interpret an expression *)
let interp (e:string) : unit =
  e |> parse |> eval |> extract_value


(*************************************************)

let return_value = function
    | String s -> s
    | Title t  -> t
    | _ -> failwith "Not a value"


let interp2 (e:string) : string =
  e |> parse |> eval |> return_value
```

```
(************************************************)

let rec loop () =
  let i = read_line ()
  in
    if i = "q" then
    let () = print_string "Bye!"
    in print_newline ()

  else
    let () = printf "%s\n" (interp2 i)
    in loop ();;


exception Die;;
let sod = function
    Some v -> v
  | None -> raise Die;;


class interpreter_handler =
object (self)
  inherit Interpreter.iface
  method interpret code  = interp2 (sod code)
end


let doserver () =
  let h = new interpreter_handler in
  let proc = new Interpreter.processor h in
  let port = 9090 in
  let pf = new TBinaryProtocol.factory in
  let server = new TThreadedServer.t
         proc
         (new TServerSocket.t port)
         (new Transport.factory)
         pf
         pf
  in
    server#serve
```

```
;;

(* let () = print_string "type q for quit): \n"; loop () *)

(* print_string "Starting sever...\n"; flush stdout; printf "%s\n" ("
   ↪ application" |> parse |> eval ) *)

doserver();;
```

5.4.3 Integração entre cliente e controlador Floodlight

Esse código demonstra como o cliente do *Apache Thrift* foi integrado ao módulo do controlador OpenFlow Floodlight, que encaminha mensagens de serviço entre entidades e o DTS, mensagens que são parte da interface definida pelo Interfacer para intermediar o serviço de título e as entidades. O *Apache Thrift* gera códigos em diversas linguagens de programação baseados em uma definição de interface de serviço. As requisições de entidades e respostas do DTS são trocadas através dessa interface de serviço.

Isso permite que uma solução seja implementada usando linguagens de programação diferentes e mais apropriadas a cada tarefa. O controlador Floodlight, por exemplo, é desenvolvido em linguagem de programação Java. Assim, foi gerada uma biblioteca contendo o cliente *Apache Thrift* e a interface de serviços implementada em Java.

Listing 5.3 – Integração lado Floodlight

```
package net.floodlightcontroller.etarch;

import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Map;

import net.floodlightcontroller.core.module.FloodlightModuleContext;
import net.floodlightcontroller.core.module.FloodlightModuleException
   ↪ ;
import net.floodlightcontroller.core.module.IFloodlightModule;
import net.floodlightcontroller.core.module.IFloodlightService;
import net.floodlightcontroller.packet.*;
import net.floodlightcontroller.core.FloodlightContext;
import net.floodlightcontroller.core.IFloodlightProviderService;
import net.floodlightcontroller.core.IOFMessageListener;
import net.floodlightcontroller.core.IOFSwitch;
```

```
import org.apache.thrift.TException;
import org.apache.thrift.protocol.TBinaryProtocol;
import org.apache.thrift.protocol.TProtocol;
import org.apache.thrift.transport.TFramedTransport;
import org.apache.thrift.transport.TSocket;
import org.apache.thrift.transport.TTransport;
import net.floodlightcontroller.etarch.xpsa.thrift.*;

import org.projectfloodlight.openflow.protocol.OFMessage;
import org.projectfloodlight.openflow.protocol.OFType;
import org.projectfloodlight.openflow.types.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class DTSAModule implements IFloodlightModule,
    ↪ IOFMessageListener {
  protected IFloodlightProviderService floodlightProvider;
  protected static Logger logger;


  @Override
  public Collection<Class<? extends IFloodlightService>>
     ↪ getModuleServices() {
    // TODO Auto-generated method stub
    return null;
  }

  @Override
  public Map<Class<? extends IFloodlightService>, IFloodlightService>
     ↪  getServiceImpls() {
    // TODO Auto-generated method stub
    return null;
  }

  @Override
  public Collection<Class<? extends IFloodlightService>>
     ↪ getModuleDependencies() {
    // TODO Auto-generated method stub
      Collection<Class<? extends IFloodlightService>> l =
              new ArrayList<Class<? extends IFloodlightService>>();
          l.add(IFloodlightProviderService.class);
```

```
55          return l;

    }

    @Override
60  public void init(FloodlightModuleContext context) throws
       ↪ FloodlightModuleException {
      // TODO Auto-generated method stub
        floodlightProvider = context.getServiceImpl(
           ↪ IFloodlightProviderService.class);
        logger = LoggerFactory.getLogger(DTSAModule.class);

65  }

    @Override
    public void startUp(FloodlightModuleContext context) throws
       ↪ FloodlightModuleException {
      // TODO Auto-generated method stub
70    floodlightProvider.addOFMessageListener(OFType.PACKET_IN, this)
         ↪ ;

    }

    @Override
75  public String getName() {
      // TODO Auto-generated method stub
        return DTSAModule.class.getSimpleName();
    }

80  @Override
    public boolean isCallbackOrderingPrereq(OFType type, String name) {
      // TODO Auto-generated method stub
      return false;
    }
85
    @Override
    public boolean isCallbackOrderingPostreq(OFType type, String name)
       ↪ {
      // TODO Auto-generated method stub
      return false;
90  }
```

```
@Override
public Command receive(IOFSwitch sw, OFMessage msg,
    ↪ FloodlightContext cntx) {
 switch (msg.getType()) {
 case PACKET_IN:
     /* Retrieve the deserialized packet in message */
     Ethernet eth = IFloodlightProviderService.bcStore.get(cntx,
         ↪  IFloodlightProviderService.CONTEXT_PI_PAYLOAD);

     /* Various getters and setters are exposed in Ethernet */
     MacAddress srcMac = eth.getSourceMACAddress();
     VlanVid vlanId = VlanVid.ofVlan(eth.getVlanID());

     /*
      * Check the ethertype of the Ethernet frame and retrieve
         ↪ the appropriate payload.
      * Note the shallow equality check. EthType caches and
         ↪ reuses instances for valid types.
      */
     if (eth.getEtherType() == EthType.IPv4) {
         /* We got an IPv4 packet; get the payload from Ethernet
             ↪  */
         IPv4 ipv4 = (IPv4) eth.getPayload();

         /* Various getters and setters are exposed in IPv4 */
         byte[] ipOptions = ipv4.getOptions();
         IPv4Address dstIp = ipv4.getDestinationAddress();

         /*
          * Check the IP protocol version of the IPv4 packet's
             ↪ payload.
          */
         if (ipv4.getProtocol() == IpProtocol.TCP) {
             /* We got a TCP packet; get the payload from IPv4
                 ↪ */
             TCP tcp = (TCP) ipv4.getPayload();
             logger.info("IPv4␣TCP␣packet:␣{}", tcp.toString());

             /* Various getters and setters are exposed in TCP
                 ↪ */
             TransportPort srcPort = tcp.getSourcePort();
             TransportPort dstPort = tcp.getDestinationPort();
```

```
                short flags = tcp.getFlags();

                /* Your logic here! */
            } else if (ipv4.getProtocol() == IpProtocol.UDP) {
                /* We got a UDP packet; get the payload from IPv4
                   ↪ */
                UDP udp = (UDP) ipv4.getPayload();
                logger.info("IPv4 UDP packet: {}", udp.toString());
                /* Various getters and setters are exposed in UDP
                   ↪ */
                TransportPort srcPort = udp.getSourcePort();
                TransportPort dstPort = udp.getDestinationPort();

                /* Your logic here! */
            }

        } else if (eth.getEtherType() == EthType.IPv6) {
            IPv6 ipv6 = (IPv6) eth.getPayload();
            logger.info("IPv6 packet: {}", ipv6.toString());



    } else if (eth.getEtherType() == EthType.ARP) {
            /* We got an ARP packet; get the payload from Ethernet
               ↪ */
            ARP arp = (ARP) eth.getPayload();
            logger.info("ARP packet: {}", arp.toString());

            /* Various getters and setters are exposed in ARP */
            boolean gratuitous = arp.isGratuitous();

        } else {
            /* Unhandled ethertype */
            EthType et = eth.getEtherType();
            Data ukn = (Data) eth.getPayload();
            String s = new String(ukn.getData(), StandardCharsets.
               ↪ UTF_8);
            logger.info("Unknown EtherType: {} on packet received
               ↪ with data: {}", et.toString(), s);
    try {
```

```
                TTransport transport;
                transport = new TSocket("localhost", 9090);
                transport.open();


                TProtocol protocol = new  TBinaryProtocol(transport);
                Interpreter.Client client = new Interpreter.Client(
                    ↪ protocol);
                String response = client.interpret(s);

                logger.info("Response␣from␣interpreter␣server:␣{}",
                    ↪ response);
                transport.close();

            } catch (TException x) {
                x.printStackTrace();
              }

            }
            break;
        default:
            break;
        }
        return Command.CONTINUE;
    }


}
```

5.5 Análise dos Resultados

Entende-se que os principais resultados desta tese foram apresentados no Capítulo 4 e Capítulo 5. O método de especificação de interfaces, que associa técnicas de modelagem conceitual orientada a ontologia com um processo de desenvolvimento de software orientado a modelos, e uma especificação do MTE na linguagem de modelagem OntoUML.

O método adota OntoUML como um mecanismo de especificação do MTE, o que lhe assegura alguns benefícios, como maior expressividade, devido a teoria ontológica UFO que suporta a OntoUML e lhe confere uma semântica mais rigorosa, principalmente se comparada à OWL. A melhoria de extensibilidade, que permite adicionar aspectos que não foram modelados a princípio, como segurança, por exemplo, não foi experimentada nesta tese.

Enquanto OWL é otimizada para a construção de bases de fatos que devem ser submetidas a mecanismos de raciocínio e inferência afim de obter um conjunto de deduções, OntoUML está mais alinhada ao propósito de representação de conhecimento sobre especificações de arquitetura de rede.

As ferramentas de avaliação do editor de ontologias Menthor (MOREIRA et al., 2016) garante a corretude sintática dos modelos através da verificação de sintaxe, colabora na redução de erros de interpretação através de simulações e na melhoria dos modelos através da busca de anti-padrões. No entanto, isso não garante corretude semântica ao modelo MTE, tão pouco lhe confere o caráter de ontologia compartilhada, cuja importância é apontada por (GUARINO; OBERLE; STAAB, 2009).

A infra-estrutura de desenvolvimento orientada a modelos instanciada no método é baseada na arquitetura de desenvolvimento orientada a modelos EMF, propiciando uma integração natural com as implementações do meta-modelo OntoUML e do editor de OntoUML, incorporado ao Menthor, que também foram desenvolvidas no arcabouço da EMF. O método adotou o *plugin Acceleo* da IDE Eclipse como um assistente para confeccionar transformadas capazes de materializar uma implementação da ETArch mais alinhada com as boas praticas arquiteturais SDN.

Embora o método também possa ser aplicado a arquiteturas NFV, entre outras, nesta tese, optou-se por um estudo de caso bem referenciado (arquitetura ETArch) e, portanto, não houve um caso envolvendo a tecnologia NFV.

Capítulo 6

Conclusão

O resultado principal dessa investigação é uso de ontologia para representação conhecimento com o intuito de especificar interfaces de gerenciamento de rede com uma semântica de comunicação mais forte. Além disso, demonstrou-se como integrar os ambientes de modelagem e de desenvolvimento a fim de automatizar o desenvolvimento de interfaces de gerenciamento de rede. Isso foi realizado especificando uma ontologia para representar conhecimento sobre a interface de gerenciamento de uma rede experimental criada em um pesquisa sobre Internet do Futuro. Essa rede de ensaio compreende um modelo e uma arquitetura conhecidos como MTE/ETArch, cujo o objetivo era investigar como uma rede pode entender e se adaptar às necessidades das entidades comunicantes.

Identificou-se que as necessidades das entidades comunicantes farão mais sentido para a rede, mais especificamente para os gerentes de rede, se elas forem descritas em função da operação da rede. Por exemplo, suponha que a necessidade de uma entidade seja evitar atrasos no transporte de seus dados. Pode-se descrever esse requisito de comunicação em função de propriedades, ou operações da rede. Ao definir atraso como a média da soma das funções da carga de processamento, tamanho da fila, velocidade de transmissão e tempo de propagação, o fator mais importante é ter uma definição compartilhada do que vem a ser atraso, ao invés da definição de atraso em si. A MTE/ETArch empregou OWL para entregar essa conceitualização compartilhada, não só do que vem a ser atraso, mas de qualquer entidade e suas respectivas necessidades. Visto que OWL não pode escalar semanticamente, ela foi substituída neste trabalho pela UFO.

Os primeiros esforços nesse sentido podem ser rastreados até a definição do *Abstract Syntax Notation One* (ASN.1) e sua variante específica para a Internet, o SMI. Esses esforços focam em garantir que a estrutura das mensagens de gerenciamento trocadas mantenham um padrão. Essa padronização no nível da sintaxe era suficiente devido ao alto grau de intervenção humana, isto é, de intervenção dos gerentes, no gerenciamento da rede. Em outras palavras, garantindo que a rede transmita, armazene e processe as informações de gerenciamento de maneira estrutura, permite aos gerentes realizar a interpretação, pode-se dizer, a avaliação semântica dessas informações. Portanto, a parte

de monitoramento da rede estava resolvida.

Outro aspecto, é o gerenciamento da rede no outro sentido, ou seja a configuração da rede. Essa questão foi resolvida também no nível da sintaxe, através da definição de gramáticas livres de contexto reconhecendo linguagens artificiais específicas para gerenciar diversos dispositivos e contextos da rede. Uma vez que o gerente esteja treinado na sintaxe de configuração de determinado componente e contexto da rede ele será capaz de interpretar, enfatizando novamente o caráter semântico, eventuais problemas de rede e traduzir a solução para a linguagem do respectivo componente, sanando assim o problema.

Para justificar a necessidade de interfaces mais semânticas, apresenta-se o argumento de que softwarização da rede tem tornado a qualidade e a quantidade dessas interpretações e intervenções que o gerente deve realizar para manter a rede funcionando, mais intensas. Com uma quantidade de usuários, dados e componentes e serviços crescente a rede caminha em direção ao limite em que o gerente de intervir com a qualidade e agilidade desejáveis, por exemplo, necessidade de aprender varias linguagens artificiais, sendo que, implementar uma configuração de rede pode demandar o uso de várias simultaneamente. Para agravar a situação, o comportamento dinâmico da rede requer configurações cada vez mais automatizadas e sincronizadas.

Esta teste se relaciona com esses esforços dos primórdios e com outras pesquisas de sintaxe para a comunicação de gerenciamento mais recentes, como YANG e o uso de de AI/ML para automatizar o gerenciamento de rede. O poder da AI é diretamente proporcional ao arcabouço semântico usado. Muito se fala da capacidade de processar imensas quantidades de dados e do aprendizado através deste processo. No entanto, como representar o conhecimento adquirido via tais métodos? Como armazenar esse conhecimento e reutilizá-lo de maneira barata e simples? A principal contribuição deste trabalho está em fomentar interfaces semanticamente enriquecidas para aliviar o fardo de gerenciar estruturas complexas, o que vai na direção inversa dos trabalhos anteriores, que visam uma estruturação precisa da informação para aproveitar o potencial humano de avaliação semântica.

Em outras palavras, propõe-se que os componentes da rede passem a usar uma linguagem artificial com aspectos semelhantes ao da linguagem natural. Com o objetivo de remover do gerente o peso de ficar traduzindo da conceitualização completa da rede para conceitualizações específicas de cada componente. Ao mapear essas conceitualizações e dominar as linguagens que as representam um gerente pode vislumbrar quais configurações são adequadas a determinado problema. Elevando o nível semântico da rede como um todo para permitir um encadeamento semântico mais direto desde o nível hardware até o mais alto nível de abstração deve tornar este mapa de conceitualizações muito mais simples e levar a uma quantidade de linguagens, embora mais complexas, muito menor. Defende-se que uma vez despendido o esforço de aprender essas linguagens, o esforço de gerenciamento será menor e a rede será mais flexível.

Neste ponto, é necessário admitir, que embora a ontologia seja muito útil como ferramenta para auxiliar na escolha de símbolos apropriados para determinado objeto, isto é, no mapeamento entre palavras e significados. Esta semântica está restrita ao nível léxico (das palavras), por exemplo, um componente de rede que realiza o controle se tornar conhecido como Controlador, ou um componente que lida com interfaces ficar conhecido como Interfaceador. Ontologia deve ser usada para representar este mapeamento entre a palavra e seu significado. Entretanto, para alcançar o calculo semântico de sentenças, são necessárias não apenas regras sintáticas, mas também regras sobre quais operações se aplicam a depender da estrutura sintática.

Pensava-se que, com o compromisso ontológico herdado da UFO, uma ontologia de domínio construída com universais da UFO seria suficiente para garantir a precisão semântica de uma determinada linguagem formal L, descrita por uma gramatica livre de contexto G. De fato, este estudo introduziu de uma semântica mais forte ao substituir a OWL pela UFO, apesar de que, como um subconjunto da lógica de descrição, OWL está amparada por um arcabouço muito mais sólido. Ao comparar a força semântica da OWL e da UFO com base em teorias matemáticas subjacentes, OWL está quase no nível de Lógica de Descrição, enquanto que a UFO está calcada na Lógica Modal (SPLENDIANI et al., 2011).

Assim, sabe-se que além da gramática G, também será necessária uma semântica. Tomando o livro (WINSKEL, 1993) como exemplo, constata-se que existem diversos tipos de semântica como a, operacional, denotacional e axiomática, citar algumas. Supeita-se, que temas recorrentes nas pesquisas de projeto e gerenciamento de rede tenham forte ligação, isto é, que eles podem ser formalmente relacionados uma vez que haja uma teoria adequada. Abstrações, interfaces, intenção e politica podem ser informalmente definidos como um mapeamento entre esses símbolos e significados que não são concretos, não são palpáveis, que existem apenas na conceitualização da comunidade que os utiliza.

Este trabalho apresentou o Interfacer, um método de especificação de interfaces para Internet do Futuro, que associa a teoria de modelagem conceitual orientada a ontologia UFO e sua respectiva linguagem de modelagem OntoUML com o processo de desenvolvimento de software orientado a modelos da EMF.À medida que aumenta a adoção de tecnologias de software nas redes, multiplicam-se os atores (desenvolvedores, provedores de serviço, operadores de rede) envolvidos nos processos de projeto, desenvolvimento e implantação de arquiteturas e serviços de rede, que se reflete em mais custos e atrasos para a entrega de produtos e serviços. Portanto, a capacidade de especificar, desenvolver e integrar interfaces específicas para serviços representa um ganho no processo de projeto, desenvolvimento e implantação de redes definidas por software.

Esse método corrobora a ideia de que o desenvolvimento de software orientado a modelos de alta expressividade, como os derivados da modelagem conceitual orientada a ontologia, induz a ganhos de produtividade ao fomentar uma linguagem mais precisa, e

ao criar artefatos associados com uma conceitualização única ao longo do processo produtivo. Portanto, afirma-se que o uso de ontologias de referencia contribui para estabelecer uma conceitualização compartilhada, que servirá de base para a aplicação de semânticas formais no desenvolvimento de interfaces de gerenciamento de rede com semântica enriquecida. Além disso, o foco no aspecto arquitetural de interfaces contribui com a codificação automatizada dessas interfaces, aumentando a interoperabilidade das soluções.

6.1 Principais Contribuições

Resumindo, este trabalho contribuiu com a melhoria dos processos de projeto, desenvolvimento e implantação de arquiteturas de comunicação especificando interfaces com semântica mais forte para o ambiente de rede MTE/ETArch:

- Uma ontologia para representar o conhecimento sobre os principais conceitos da interface de gerenciamento do MTE/ETArch foi especificada em linguagem de modelagem conceitual OntoUML, contendo três diagramas que contemplam os conceitos de Entidade, Dispositivo, e Hospedagem. E diretrizes foram traçadas para a especificação do conceito de *Workspace*;

- Demonstrou-se como integrar as infra-estruturas de modelagem e de desenvolvimento orientado a modelos usando o editor de ontologias Menthor, a versão de modelagem do IDE Eclipse, o meta-modelo OntoUML e o *plugin* de transformação *Acceleo* a fim de automatizar o desenvolvimento de interfaces de gerenciamento semântico;

6.2 Trabalhos Futuros

Recomenda-se investigar como automatizar a transformação do modelo OntoUML para especificações para geradores de analisadores léxicos e sintáticos. O analisador sintático deve referenciar certos nós da arvore de universais do modelo, por exemplo, entidade, no caso do MTE/ETArch. Via de regra, esses nos devem estar associados a classes morfológicas abertas, como os substantivos e os verbos. A ideia é que novos objetos e ações possam ser adicionados e sejam acessados através da interface usando sentenças de mesma estrutura sintática. Em alguns casos os filhos desses nos referenciados podem se comportar como qualificadores, como se fossem adjetivos.

A adoção de um nível semântico mais elevado nas interfaces de gerenciamento de rede trás muitos de desafios, no sentido que apenas teorias completas e corretas podem produzir interfaces de gerenciamento prontas para um ambiente de produção. Muito ainda deve ser investigado a respeito das teorias semânticas como a operacional, denotacional e

axiológica, a fim de entender como elas se complementam e como elas podem trabalhar em conjunto com a UFO para prover uma semântica formal melhorada.

Outra preocupação é em investigar o custo, tanto em termos de complexidade, quanto em termos de recursos para introduzir essas teorias nas redes softwarizadas, versus os benefícios dessa elevação na precisão semântica das interfaces de gerenciamento. No caso dessas pesquisas mostrarem alguma viabilidade, vislumbra-se investigações adicionais para sondar como essa semântica pode ser integrada a plataformas de AI e ML de modo a auxiliar no armazenamento do conhecimento adquirido, na automação da operação e segurança da rede.

6.3 Contribuições em Produção Bibliográfica

Um artigo entitulado *Interfacer: A Model-Driven Development Method for SDN Applications* descreve o método de especificação de interfaces com processo de desenvolvimento orientado a modelos para redes definidas por software SDN foi aceito para publicação na conferencia AINA: *Advanced Information Networking and Applications* (SILVA et al., 2020).

Outro artigo entitulado *Interfacer: Towards Customizable Services in Network Softwarization*, discute as implicações da aplicação do método Interfacer na produção de implementações mais aderentes à arquitetura SDN como um passo em direção a serviços de rede customizáveis foi submetido ao periódico *Security and Communication Networks*

Referências

AHLGREN, B.; ARKKO, J.; EGGERT, L.; RAJAHALME, J. A node identity internetworking architecture. In: IEEE. **Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications**. 2006. p. 1–6. Disponível em: <https://doi.org/10.1109/INFOCOM.2006.51>.

ATKINSON, R.; BHATTI, S. **RFC 6740 Identifier-locator network protocol (ILNP) architectural description**. 2012. Disponível em: <https://tools.ietf.org/rfc/rfc6740.txt>.

BRAGA, B. F.; ALMEIDA, J. P. A.; GUIZZARDI, G.; BENEVIDES, A. B. Transforming OntoUML into Alloy: Towards conceptual model validation using a lightweight formal method. **Innovations in Systems and Software Engineering**, v. 6, n. 1, p. 55–63, 2010. ISSN 16145046. Disponível em: <http://doi.org/10.1007/s11334-009-0120-5>.

BUSH, R.; MEYER, D. **RFC 3439 Some Internet Architectural Guidelines and Philosophy**. 2002. 1-28 p. Disponível em: <http://tools.ietf.org/rfc/rfc3439.txt>.

CASADO, M.; FOSTER, N.; GUHA, A. Abstractions for software-defined networks. **Communications of the ACM**, ACM, New York, NY, USA, v. 57, n. 10, p. 86–95, 9 2014. ISSN 15577317. Disponível em: <http://doi.acm.org/10.1145/2661061.2661063>.

CASADO, M.; KOPONEN, T.; SHENKER, S.; TOOTOONCHIAN, A. Fabric: A retrospective on evolving SDN. In: **HotSDN'12 - Proceedings of the 1st ACM International Workshop on Hot Topics in Software Defined Networks**. New York, NY, USA: ACM, 2012. (HotSDN '12), p. 85–89. ISBN 9781450314770. Disponível em: <http://doi.acm.org/10.1145/2342441.2342459>.

DEERING, S. E. **RFC 2460 Internet protocol, version 6 (IPv6) specification**. 1998. Disponível em: <http://tools.ietf.org/rfc/rfc2460.txt>.

FARINACCI, D.; LEWIS, D.; MEYER, D.; FULLER, V. **RFC 6830 The locator/ID separation protocol (LISP)**. 2013. Disponível em: <http://tools.ietf.org/rfc/rfc6830.txt>.

FISHER, D. US National Science foundation and the future internet design. **Computer Communication Review**, ACM New York, NY, USA, v. 37, n. 3, p. 85–87, 2007. ISSN 01464833. Disponível em: <https://doi.org/10.1145/1273445.1273459>.

GALIS, A.; CLAYMAN, S.; LEFEVRE, L.; FISCHER, A.; MEER, H. D.; RUBIO-LOYOLA, J.; SERRAT, J.; DAVY, S. **The Future Internet**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. v. 6656. 19–33 p. (Lecture Notes in Computer Science, v. 6656). ISBN 978-3-642-20897-3. Disponível em: <http://dx.doi.org/10.1007/978-3-642-20898-0>.

GAVRAS, A.; KARILA, A.; FDIDA, S.; MAY, M.; POTTS, M. Future internet research and experimentation: The FIRE initiative. **Computer Communication Review**, ACM, New York, NY, USA, v. 37, n. 3, p. 89–92, 7 2007. ISSN 01464833. Disponível em: <http://doi.acm.org/10.1145/1273445.1273460>.

GUARINO, N.; OBERLE, D.; STAAB, S. What Is an Ontology? In: STAAB, S.; STUDER, R. (Ed.). **Handbook on Ontologies**. Second. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 1–17. Disponível em: <http://link.springer.com/10.1007/978-3-540-92673-3_0>.

GUERSON, J.; SALES, T. P.; GUIZZARDI, G.; ALMEIDA, J. P. A. OntoUML lightweight editor: A model-based environment to build, evaluate and implement reference ontologies. In: **Proceedings of the 2015 IEEE 19th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations, EDOCW 2015**. Washington, DC, USA: IEEE Computer Society, 2015. (EDOCW '15), p. 144–147. ISBN 9781467393317. ISSN 2325-6583. Disponível em: <https://doi.org/10.1109/EDOCW.2015.17>.

GUIZZARDI, G. **Ontological Foundations for Structural Conceptual Models**. [s.n.], 2005. ISBN 9075176813. Disponível em: <https://research.utwente.nl/en/publications/ontological-foundations-for-structural-conceptual-models/>.

GUIZZARDI, G.; FALBO, R. A.; GUIZZARDI, R. S. A importância de ontologias de fundamentação para a engenharia de ontologias de domínio: o caso do domínio de processos de software. **IEEE Latin America Transactions**, v. 6, n. 3, p. 244–251, 7 2008. ISSN 15480992. Disponível em: <http://doi.org/10.1109/TLA.2008.4653854>.

JÚNIOR, J. C.; PRUDÊNCIO, A. C.; WILLRICH, R.; SILVA, M. P. da. A semantic approach for QoS specification of communication services using QoE parameters. **Journal of the Brazilian Computer Society**, v. 19, n. 3, p. 207–221, 9 2013. ISSN 01046500. Disponível em: <https://doi.org/10.1007/s13173-012-0094-2>.

LEMA, J. C.; SILVA, F.; NETO, A.; SILVA, F. D. O.; FROSI, P.; GUIMARES, C.; CORUJO, D.; AGUIAR, R. Evolving future Internet clean-slate Entity Title Architecture with quality-oriented control plane extensions. In: **Advanced International Conference on Telecommunications, AICT**. International Academy, Research and Industry Association, IARIA, 2014. v. 2014-July, n. July, p. 161–167. ISSN 23084030. Disponível em: <https://repositorio.ufrn.br/handle/123456789/18107>.

MASOLO, C.; GUIZZARDI, G.; VIEU, L.; BOTTAZZI, E.; FERRARIO, R. Relational roles and qua-individuals. In: **Hyatt Crystal City**. AAAI Press, 2005. p. 3–6. Disponível em: <https://www.aaai.org/Papers/Symposia/Fall/2005/FS-05-08/FS05-08-017.pdf>.

MILI, H.; VALTCHEV, P.; SZATHMARY, L.; BOUBAKER, A.; LESHOB, A.; CHARIF, Y.; MARTIN, L. Ontology-based model-driven development of a destination management portal: Experience and lessons learned. **Software:**

Practice and Experience, v. 48, n. 8, p. 1438–1460, 2018. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2581>.

MOREIRA, J. J. L.; SALES, T. P.; GUERSON, J.; BRAGA, B. F. B.; BRASILEIRO, F.; SOBRAL, V. M.; SALES, T. P.; GUERSON, J.; BRAGA, B. F. B.; BRASILEIRO, F.; SOBRAL, V. **Menthor Editor: an ontology-driven conceptual modeling platform**. [S.l.], 2016. v. 1660. Disponível em: <http://ceur-ws.org/Vol-1660/>.

NIKANDER, P.; GURTOV, A.; HENDERSON, T. R. Host identity protocol (hip): Connectivity, mobility, multi-homing, security, and privacy over ipv4 and ipv6 networks. **IEEE Communications Surveys & Tutorials**, IEEE, v. 12, n. 2, p. 186–204, 2010. Disponível em: <https://doi.org/10.1109/SURV.2010.021110.00070>.

PEREIRA, J. H. d. S. **Modelo de título para a próxima geração de Internet**. Tese (Doutorado) — Universidade de São Paulo, 02 2012. Disponível em: <https://www.teses.usp.br/teses/disponiveis/3/3142/tde-13062013-113304/pt-br.php>.

PEREIRA, J. H. D. S.; KOFUJI, S. T.; ROSA, P. F. Horizontal address ontology in internet architecture. In: AGHA, K. A.; BADRA, M.; NEWBY, G. B. (Ed.). **3rd International Conference on New Technologies, Mobility and Security, NTMS 2009**. IEEE, 2009. p. 1–6. ISBN 9781424462735. Disponível em: <http://dx.doi.org/10.1109/NTMS.2009.5384801>.

PEREIRA, J. H. D. S.; SILVA, F. D. O.; FILHO, E. L.; KOFUJI, S. T.; ROSA, P. F. Title model ontology for future internet networks. In: DOMINGUE, J.; GALIS, A.; GAVRAS, A.; ZAHARIADIS, T. B.; LAMBERT, D.; CLEARY, F.; DARAS, P.; KRCO, S.; MÜLLER, H.; LI, M.-S.; SCHAFFERS, H.; LOTZ, V.; ALVAREZ, F.; STILLER, B.; KARNOUSKOS, S.; AVESSTA, S.; NILSSON, M. (Ed.). **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**. Berlin, Heidelberg: Springer, 2011. (Lecture Notes in Computer Science, v. 6656), p. 103–114. ISBN 9783642208973. ISSN 03029743. Disponível em: <http://dx.doi.org/10.1007/978-3-642-20898-0_8>.

PEREIRA, J. H. de S.; SILVA, F. O.; FILHO, E. L.; KOFUJI, S. T.; ROSA, P. F. Title model ontology for future internet networks. In: DOMINGUE, J.; GALIS, A.; GAVRAS, A.; ZAHARIADIS, T. B.; LAMBERT, D.; CLEARY, F.; DARAS, P.; KRCO, S.; MÜLLER, H.; LI, M.; SCHAFFERS, H.; LOTZ, V.; ALVAREZ, F.; STILLER, B.; KARNOUSKOS, S.; AVESSTA, S.; NILSSON, M. (Ed.). **The Future Internet - Future Internet Assembly 2011: Achievements and Technological Promises**. Springer, 2011. (Lecture Notes in Computer Science, v. 6656), p. 103–116. Disponível em: <http://dx.doi.org/10.1007/978-3-642-20898-0_8>.

PERGL, R.; SALES, T. P.; RYBOLA, Z. Towards OntoUML for Software Engineering: From Domain Ontology to Implementation Model. In: CUZZOCREA, A.; MAABOUT, S. (Ed.). **Model and Data Engineering**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 249–263. ISBN 978-3-642-41366-7. Disponível em: <https://doi.org/10.1007/978-3-642-41366-7_21>.

PETERSON, L.; ANDERSON, T.; BLUMENTHAL, D.; CASEY, D.; CLARK, D.; ESTRIN, D.; EVANS, J.; RAYCHAUDHURI, D.; REITER, M.; REXFORD, J.; others. GENI design principles. 2006. Disponível em: <https://doi.org/10.1109/MC.2006.307>.

POSTEL, J. **RFC 760: DoD standard Internet Protocol**. 1980. Disponível em: <https://tools.ietf.org/rfc/rfc760.txt>.

_____. **RFC 761: DoD standard Transmission Control Protocol**. 1980. Disponível em: <https://tools.ietf.org/rfc/rfc761.txt>.

_____. **RFC 825: Request for comments on Requests For Comments**. 1982. Disponível em: <https://tools.ietf.org/rfc/rfc825.txt>.

QUOITIN, B.; IANNONE, L.; LAUNOIS, C. de; BONAVENTURE, O. Evaluating the benefits of the locator/identifier separation. In: **Proceedings of 2nd ACM/IEEE International Workshop on Mobility in the Evolving Internet Architecture**. New York, NY, USA: ACM, 2007. (MobiArch '07), p. 5:1–5:6. ISBN 978-1-59593-784-1. Disponível em: <http://doi.acm.org/10.1145/1366919.1366926>.

SALES, T. P.; SALES, T. P. **Ontology Validation for Managers**. Tese (Doutorado) — Universidade Federal do Espírito Santo, 2014. Disponível em: <http://repositorio.ufes.br/handle/10/4273>.

SALTZER, J. **RFC 1498: On the Naming and Binding of Network Destinations**. 1993. Disponível em: <https://tools.ietf.org/rfc/rfc1498.txt>.

SANTOS, E. D. S.; PEREIRA, F. S. F.; PEREIRA, J. H.; THEODORO, L. C.; ROSA, P. F.; KOFUJI, S. T.; PEREIRA, J. H. de S.; THEODORO, L. C.; ROSA, P. F.; KOFUJI, S. T. Meeting Services and Networks in the Future Internet. In: DOMINGUE, J.; GALIS, A.; GAVRAS, A.; ZAHARIADIS, T. B.; LAMBERT, D.; CLEARY, F.; DARAS, P.; KRCO, S.; MÜLLER, H.; LI, M.-S.; SCHAFFERS, H.; LOTZ, V.; ALVAREZ, F.; STILLER, B.; KARNOUSKOS, S.; AVESSTA, S.; NILSSON, M. (Ed.). **Future Internet Assembly**. Springer, 2011. (Lecture Notes in Computer Science, v. 6656), p. 339–350. ISBN 978-3-642-20897-3. ISSN 03029743. Disponível em: <http://dx.doi.org/10.1007/978-3-642-20898-0_24>.

SHENKER, S. Fundamental Design Issues for the Future Internet. **IEEE J.Sel. A. Commun.**, IEEE Press, Piscataway, NJ, USA, v. 13, n. 7, p. 1176–1188, 9 2006. ISSN 0733-8716. Disponível em: <http://dx.doi.org/10.1109/49.414637>.

SILVA, D. O.; PEREIRA, D. S.; ROSA, P. F.; JÚNIOR, J. E. P.; SILVA, F. de O.; PEREIRA, J. H. de S.; ROSA, P. F.; EURÍPEDES, J.; JÚNIOR, P.; SILVA, F. D. O.; SOUZA, J. H. D.; ROSA, P. F.; HENRIQUE, J.; PEREIRA, D. S.; ROSA, F. Interfacer: A Model-Driven Development Method for SDN Applications. **Advanced Information Networking and Applications**, Springer International Publishing, Cham, n. Mdd, p. 643–654, 2020. Disponível em: <https://doi.org/10.1007/978-3-030-15032-7_54>.

SILVA, F. d. O. **Endereçamento por título: uma forma de encaminhamento multicast para a próxima geração de redes de computadores**. Tese (Doutorado) — Universidade de São Paulo, 10 2013. Disponível em: <http://www.teses.usp.br/teses/disponiveis/3/3142/tde-22092014-111409/pt-br.php>.

SILVA, F. de O.; DIAS, A.; FERREIRA, C. C.; SANTOS, E. D. S.; PEREIRA, F. S. F.; ANDRADE, I. C. de; PEREIRA, J. a. H. de S.; CAMARGOS, L. J.; THEODORO, L. C.; GONçALVES, M. A.; PASQUINI, R.; NETO, A. J. V.; ROSA, P. F.; KOFUJI, S. T. The future internet. In: ÁLVAREZ, F.; CLEARY, F.; DARAS, P.; DOMINGUE, J.;

GALIS, A. (Ed.). Berlin, Heidelberg: Springer-Verlag, 2012. cap. Semantically Enriched Services to Understand the Need of Entities, p. 142–153. ISBN 978-3-642-30240-4. Disponível em: <http://dl.acm.org/citation.cfm?id=2340856.2340872>.

SOUSA, N. F. Saraiva de; PEREZ, D. A. L.; ROSA, R. V.; SANTOS, M. A.; ROTHENBERG, C. E. Network Service Orchestration: A survey. **Computer Communications**, v. 142-143, p. 69–94, 2019. ISSN 1873703X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0140366418309502>.

SPLENDIANI, A.; BURGER, A.; PASCHKE, A.; ROMANO, P.; MARSHALL, M. S. Biomedical semantics in the Semantic Web. **Journal of Biomedical Semantics**, Nomos Verlagsgesellschaft mbH \& Co. KG, v. 2, n. 1, p. 187–203, 2011. ISSN 20411480. Disponível em: <https://doi.org/10.5771/0943-7444-2011-3-187>.

SRISURESH, P.; EGEVANG, K. **Traditional IP network address translator (Traditional NAT)**. [S.l.], 2000.

STEINBERG, D.; BUDINSKY, F.; PATERNOSTRO, M.; MERKS, E. **EMF: Eclipse Modeling Framework 2.0**. 2nd. ed. [S.l.]: Addison-Wesley Professional, 2009. ISBN 0321331885.

TROIS, C.; FABRO, M. D. D.; BONA, L. C. E. de; MARTINELLO, M. A Survey on SDN Programming Languages: Toward a Taxonomy. **IEEE Communications Surveys Tutorials**, v. 18, n. 4, p. 2687–2712, 2016. ISSN 1553-877X. Disponível em: <https://doi.org/10.1109/COMST.2016.2553778>.

VERDONCK, M.; GAILLY, F.; CESARE, S. de; POELS, G.; CESARE, S. D.; POELS, G. Ontology-driven conceptual modeling: A systematic literature mapping and review. **Applied Ontology**, v. 10, n. 3, p. 197–227, 2015. Disponível em: <https://doi.org/10.3233/AO-150154>.

WINSKEL, G. **The formal semantics of programming languages: an introduction**. [S.l.]: MIT press, 1993.

WU, P.; CUI, Y.; WU, J.; LIU, J.; METZ, C. Transition from ipv4 to ipv6: A state-of-the-art survey. **IEEE Communications Surveys & Tutorials**, IEEE, v. 15, n. 3, p. 1407–1424, 2013. Disponível em: <https://doi.org/10.1109/SURV.2012.110112.00200>.

ZIMMERMANN, H. Osi reference model - the iso model of architecture for open systems interconnection. **IEEE Transactions on Communications**, IEEE, v. 28, n. 4, p. 425–432, April 1980. ISSN 0090-6778. Disponível em: <https://doi.org/10.1109/TCOM.1980.1094702>.

Apêndices

APÊNDICE **A**

Códigos Fonte

A.1 Linguagem da Interface (Parser e Lexer em Ocaml)

```
{
open Lexing
open Parser

exception SyntaxError of string

let next_line lexbuf =
  let pos = lexbuf.lex_curr_p in
  lexbuf.lex_curr_p <-
    { pos with pos_bol = lexbuf.lex_curr_pos;
               pos_lnum = pos.pos_lnum + 1
    }

let keyword_table = Hashtbl.create 53
let _ =
  List.iter (fun (kwd, tok) -> Hashtbl.add keyword_table kwd tok)
            [
              "application", APPLICATION;
              "content", CONTENT;
              "device", DEVICE;
              "user", USER;
              "end", END;
  ]

}
```

```
let int = '-'? ['0'-'9'] ['0'-'9']*
let digit = ['0'-'9']
let frac = '.' digit*
let exp = ['e' 'E'] ['-' '+']? digit+
let float = digit* frac? exp?

let white = [' ' '\t']+
let newline = '\r' | '\n' | "\r\n"
(*let id = ['a'-'z' 'A'-'Z' '_'] ['a'-'z' 'A'-'Z' '0'-'9' '_']* *)

rule read =
  parse
  | white    { read lexbuf }
  | newline  { next_line lexbuf; read lexbuf }
  | int      { INT (int_of_string (Lexing.lexeme lexbuf)) }
  | ['a'-'z' 'A'-'Z' '_'] ['a'-'z' 'A'-'Z' '0'-'9' '_']* as id
             { try Hashtbl.find keyword_table id with Not_found ->
               ↪ IDENT id }
  | '"'      { read_string (Buffer.create 17) lexbuf }
(*  | '{'      { L_BRACE }
  | '}'      { R_BRACE }
  | '['      { L_BRACK }
  | ']'      { R_BRACK }
  | '('      { L_PAREN }
  | ')'      { R_PAREN }
  | ':'      { COLON } *)
  | ','      { COMMA }
  | '='      { ASSIGN }
  | '|'      { OPTION }
  | '.'      { DOT }
  | _ { raise (SyntaxError ("Unexpected char: " ^ Lexing.lexeme
     ↪ lexbuf)) }
  | eof      { EOF }


and read_string buf =
  parse
  | '"'       { STRING (Buffer.contents buf) }
  | '\\' '/'  { Buffer.add_char buf '/'; read_string buf lexbuf }
  | '\\' '\\' { Buffer.add_char buf '\\'; read_string buf lexbuf }
  | '\\' 'b'  { Buffer.add_char buf '\b'; read_string buf lexbuf }
  | '\\' 'f'  { Buffer.add_char buf '\012'; read_string buf lexbuf }
```

```
  | '\\' 'n'  { Buffer.add_char buf '\n'; read_string buf lexbuf }
  | '\\' 'r'  { Buffer.add_char buf '\r'; read_string buf lexbuf }
  | '\\' 't'  { Buffer.add_char buf '\t'; read_string buf lexbuf }
  | [^ '"' '\\']+
    { Buffer.add_string buf (Lexing.lexeme lexbuf);
      read_string buf lexbuf
    }
  | _ { raise (SyntaxError ("Illegal string character: " ^ Lexing.
     ↪ lexeme lexbuf)) }
  | eof { raise (SyntaxError ("String is not terminated")) }
```

A.2 Módulo do Floodlight

```
[comment encoding = UTF-8 /]
[module FloodlightModule('OntoUML')]


[template public generateModule(aKind : Kind)]

[file (aKind.name + '.java', false, 'UTF-8')]
public [aKind.name/] implements IOFMessageListener, IFloodlightModule
   ↪ {
    @Override
    public String getName() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public boolean isCallbackOrderingPrereq(OFType type, String name)
       ↪ {
        // TODO Auto-generated method stub
        return false;
    }
}

[/file]
[/template]
```

A.3 Apache Thrift

```
(*
```

```
  Autogenerated by Thrift Compiler (0.9.0)

  DO NOT EDIT UNLESS YOU ARE SURE YOU KNOW WHAT YOU ARE DOING
*)

open Thrift
open Xpsa_types

(* HELPER FUNCTIONS AND STRUCTURES *)

class interpret_args =
object (self)
  val mutable _code : string option = None
  method get_code = _code
  method grab_code = match _code with None->raise (Field_empty "
    ↪ interpret_args.code") | Some _x0 -> _x0
  method set_code _x0 = _code <- Some _x0
  method unset_code = _code <- None
  method reset_code = _code <- None

  method copy =
      let _new = Oo.copy self in
    _new
  method write (oprot : Protocol.t) =
    oprot#writeStructBegin "interpret_args";
    (match _code with None -> () | Some _v ->
      oprot#writeFieldBegin("code",Protocol.T_STRING,1);
      oprot#writeString(_v);
      oprot#writeFieldEnd
    );
    oprot#writeFieldStop;
    oprot#writeStructEnd
end
let rec read_interpret_args (iprot : Protocol.t) =
  let _str3 = new interpret_args in
    ignore(iprot#readStructBegin);
    (try while true do
        let (_,_t4,_id5) = iprot#readFieldBegin in
        if _t4 = Protocol.T_STOP then
          raise Break
        else ();
        (match _id5 with
```

```
          | 1 -> (if _t4 = Protocol.T_STRING then
              _str3#set_code iprot#readString
            else
              iprot#skip _t4)
          | _ -> iprot#skip _t4);
        iprot#readFieldEnd;
      done; ()
    with Break -> ());
    iprot#readStructEnd;
    _str3

class interpret_result =
object (self)
  val mutable _success : string option = None
  method get_success = _success
  method grab_success = match _success with None->raise (Field_empty
    ↪ "interpret_result.success") | Some _x6 -> _x6
  method set_success _x6 = _success <- Some _x6
  method unset_success = _success <- None
  method reset_success = _success <- None

  method copy =
      let _new = Oo.copy self in
    _new
  method write (oprot : Protocol.t) =
    oprot#writeStructBegin "interpret_result";
    (match _success with None -> () | Some _v ->
      oprot#writeFieldBegin("success",Protocol.T_STRING,0);
      oprot#writeString(_v);
      oprot#writeFieldEnd
    );
    oprot#writeFieldStop;
    oprot#writeStructEnd
end
let rec read_interpret_result (iprot : Protocol.t) =
  let _str9 = new interpret_result in
    ignore(iprot#readStructBegin);
    (try while true do
        let (_,_t10,_id11) = iprot#readFieldBegin in
        if _t10 = Protocol.T_STOP then
          raise Break
        else ();
```

```
        (match _id11 with
          | 0 -> (if _t10 = Protocol.T_STRING then
              _str9#set_success iprot#readString
            else
              iprot#skip _t10)
          | _ -> iprot#skip _t10);
        iprot#readFieldEnd;
      done; ()
    with Break -> ());
    iprot#readStructEnd;
    _str9

class virtual iface =
object (self)
  method virtual interpret : string option -> string
end

class client (iprot : Protocol.t) (oprot : Protocol.t) =
object (self)
  val mutable seqid = 0
  method interpret code =
    self#send_interpret code;
    self#recv_interpret
  method private send_interpret code =
    oprot#writeMessageBegin ("interpret", Protocol.CALL, seqid);
    let args = new interpret_args in
      args#set_code code;
      args#write oprot;
      oprot#writeMessageEnd;
      oprot#getTransport#flush
  method private recv_interpret  =
    let (fname, mtype, rseqid) = iprot#readMessageBegin in
      (if mtype = Protocol.EXCEPTION then
        let x = Application_Exn.read iprot in
          (iprot#readMessageEnd;           raise (Application_Exn.E x
            ↪ ))
      else ());
      let result = read_interpret_result iprot in
        iprot#readMessageEnd;
        match result#get_success with Some v -> v | None -> (
          raise (Application_Exn.E (Application_Exn.create
            ↪ Application_Exn.MISSING_RESULT "interpret failed:
```

```
            ↪ unknown result")))
  end

125
  class processor (handler : iface) =
  object (self)
    inherit Processor.t

130   val processMap = Hashtbl.create 1
    method process iprot oprot =
      let (name, typ, seqid)  = iprot#readMessageBegin in
        if Hashtbl.mem processMap name then
          (Hashtbl.find processMap name) (seqid, iprot, oprot)
135       else (
          iprot#skip(Protocol.T_STRUCT);
          iprot#readMessageEnd;
          let x = Application_Exn.create Application_Exn.UNKNOWN_METHOD
             ↪  ("Unknown function "^name) in
            oprot#writeMessageBegin(name, Protocol.EXCEPTION, seqid);
140         x#write oprot;
            oprot#writeMessageEnd;
            oprot#getTransport#flush
        );
        true
145   method private process_interpret (seqid, iprot, oprot) =
      let args = read_interpret_args iprot in
        iprot#readMessageEnd;
        let result = new interpret_result in
          result#set_success (handler#interpret args#get_code);
150       oprot#writeMessageBegin ("interpret", Protocol.REPLY, seqid);
          result#write oprot;
          oprot#writeMessageEnd;
          oprot#getTransport#flush
    initializer
155   Hashtbl.add processMap "interpret" self#process_interpret;
  end


  /**
   * Autogenerated by Thrift Compiler (0.9.2)
   *
   * DO NOT EDIT UNLESS YOU ARE SURE THAT YOU KNOW WHAT YOU ARE DOING
 5 *  @generated
   */
```

```
package net.floodlight.etarch.xpsa.thrift;

import org.apache.thrift.scheme.IScheme;
import org.apache.thrift.scheme.SchemeFactory;
import org.apache.thrift.scheme.StandardScheme;

import org.apache.thrift.scheme.TupleScheme;
import org.apache.thrift.protocol.TTupleProtocol;
import org.apache.thrift.protocol.TProtocolException;
import org.apache.thrift.EncodingUtils;
import org.apache.thrift.TException;
import org.apache.thrift.async.AsyncMethodCallback;
import org.apache.thrift.server.AbstractNonblockingServer.*;
import java.util.List;
import java.util.ArrayList;
import java.util.Map;
import java.util.HashMap;
import java.util.EnumMap;
import java.util.Set;
import java.util.HashSet;
import java.util.EnumSet;
import java.util.Collections;
import java.util.BitSet;
import java.nio.ByteBuffer;
import java.util.Arrays;
import javax.annotation.Generated;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@SuppressWarnings({"cast", "rawtypes", "serial", "unchecked"})
@Generated(value = "Autogenerated by Thrift Compiler (0.9.2)", date =
    ↪ "2019-2-27")
public class Interpreter {

  public interface Iface {

    public String interpret(String code) throws org.apache.thrift.
       ↪ TException;

  }

  public interface AsyncIface {
```

```
    public void interpret(String code, org.apache.thrift.async.
        ↪ AsyncMethodCallback resultHandler) throws org.apache.thrift
        ↪ .TException;

50  }

  public static class Client extends org.apache.thrift.TServiceClient
      ↪  implements Iface {
    public static class Factory implements org.apache.thrift.
        ↪ TServiceClientFactory<Client> {
      public Factory() {}
55    public Client getClient(org.apache.thrift.protocol.TProtocol
          ↪ prot) {
        return new Client(prot);
      }
      public Client getClient(org.apache.thrift.protocol.TProtocol
          ↪ iprot, org.apache.thrift.protocol.TProtocol oprot) {
        return new Client(iprot, oprot);
60    }
    }

    public Client(org.apache.thrift.protocol.TProtocol prot)
    {
65    super(prot, prot);
    }

    public Client(org.apache.thrift.protocol.TProtocol iprot, org.
        ↪ apache.thrift.protocol.TProtocol oprot) {
      super(iprot, oprot);
70  }

    public String interpret(String code) throws org.apache.thrift.
        ↪ TException
    {
      send_interpret(code);
75    return recv_interpret();
    }

    public void send_interpret(String code) throws org.apache.thrift.
        ↪ TException
    {
```

```
80      interpret_args args = new interpret_args();
        args.setCode(code);
        sendBase("interpret", args);
      }

85    public String recv_interpret() throws org.apache.thrift.
         ↪ TException
      {
        interpret_result result = new interpret_result();
        receiveBase(result, "interpret");
        if (result.isSetSuccess()) {
90        return result.success;
        }
        throw new org.apache.thrift.TApplicationException(org.apache.
           ↪ thrift.TApplicationException.MISSING_RESULT, "interpret
           ↪ failed: unknown result");
      }

95  }
    public static class AsyncClient extends org.apache.thrift.async.
       ↪ TAsyncClient implements AsyncIface {
      public static class Factory implements org.apache.thrift.async.
         ↪ TAsyncClientFactory<AsyncClient> {
        private org.apache.thrift.async.TAsyncClientManager
           ↪ clientManager;
        private org.apache.thrift.protocol.TProtocolFactory
           ↪ protocolFactory;
100     public Factory(org.apache.thrift.async.TAsyncClientManager
           ↪ clientManager, org.apache.thrift.protocol.
           ↪ TProtocolFactory protocolFactory) {
          this.clientManager = clientManager;
          this.protocolFactory = protocolFactory;
        }
        public AsyncClient getAsyncClient(org.apache.thrift.transport.
           ↪ TNonblockingTransport transport) {
105       return new AsyncClient(protocolFactory, clientManager,
             ↪ transport);
        }
      }

      public AsyncClient(org.apache.thrift.protocol.TProtocolFactory
         ↪ protocolFactory, org.apache.thrift.async.
```

```
    ↪ TAsyncClientManager clientManager, org.apache.thrift.
    ↪ transport.TNonblockingTransport transport) {
110   super(protocolFactory, clientManager, transport);
  }

  public void interpret(String code, org.apache.thrift.async.
    ↪ AsyncMethodCallback resultHandler) throws org.apache.thrift
    ↪ .TException {
    checkReady();
115   interpret_call method_call = new interpret_call(code,
      ↪ resultHandler, this, ___protocolFactory, ___transport);
    this.___currentMethod = method_call;
    ___manager.call(method_call);
  }

120 public static class interpret_call extends org.apache.thrift.
    ↪ async.TAsyncMethodCall {
    private String code;
    public interpret_call(String code, org.apache.thrift.async.
      ↪ AsyncMethodCallback resultHandler, org.apache.thrift.
      ↪ async.TAsyncClient client, org.apache.thrift.protocol.
      ↪ TProtocolFactory protocolFactory, org.apache.thrift.
      ↪ transport.TNonblockingTransport transport) throws org.
      ↪ apache.thrift.TException {
      super(client, protocolFactory, transport, resultHandler,
        ↪ false);
      this.code = code;
125   }

    public void write_args(org.apache.thrift.protocol.TProtocol
      ↪ prot) throws org.apache.thrift.TException {
      prot.writeMessageBegin(new org.apache.thrift.protocol.
        ↪ TMessage("interpret", org.apache.thrift.protocol.
        ↪ TMessageType.CALL, 0));
      interpret_args args = new interpret_args();
130   args.setCode(code);
      args.write(prot);
      prot.writeMessageEnd();
    }

135   public String getResult() throws org.apache.thrift.TException {
      if (getState() != org.apache.thrift.async.TAsyncMethodCall.
```

```
            ↪ State.RESPONSE_READ) {
          throw new IllegalStateException("Method call not finished
            ↪ !");
        }
        org.apache.thrift.transport.TMemoryInputTransport
            ↪ memoryTransport = new org.apache.thrift.transport.
            ↪ TMemoryInputTransport(getFrameBuffer().array());
        org.apache.thrift.protocol.TProtocol prot = client.
            ↪ getProtocolFactory().getProtocol(memoryTransport);
        return (new Client(prot)).recv_interpret();
      }
    }

}

  public static class Processor<I extends Iface> extends org.apache.
     ↪ thrift.TBaseProcessor<I> implements org.apache.thrift.
     ↪ TProcessor {
    private static final Logger LOGGER = LoggerFactory.getLogger(
       ↪ Processor.class.getName());
    public Processor(I iface) {
      super(iface, getProcessMap(new HashMap<String, org.apache.
         ↪ thrift.ProcessFunction<I, ? extends org.apache.thrift.
         ↪ TBase>>()));
    }

    protected Processor(I iface, Map<String,  org.apache.thrift.
       ↪ ProcessFunction<I, ? extends  org.apache.thrift.TBase>>
       ↪ processMap) {
      super(iface, getProcessMap(processMap));
    }

    private static <I extends Iface> Map<String,  org.apache.thrift.
       ↪ ProcessFunction<I, ? extends  org.apache.thrift.TBase>>
       ↪ getProcessMap(Map<String,  org.apache.thrift.
       ↪ ProcessFunction<I, ? extends  org.apache.thrift.TBase>>
       ↪ processMap) {
      processMap.put("interpret", new interpret());
      return processMap;
    }

    public static class interpret<I extends Iface> extends org.apache
```

```
      ↪ .thrift.ProcessFunction<I, interpret_args> {
      public interpret() {
        super("interpret");
      }

      public interpret_args getEmptyArgsInstance() {
        return new interpret_args();
      }

      protected boolean isOneway() {
        return false;
      }

      public interpret_result getResult(I iface, interpret_args args)
        ↪  throws org.apache.thrift.TException {
        interpret_result result = new interpret_result();
        result.success = iface.interpret(args.code);
        return result;
      }
    }

  }

  public static class AsyncProcessor<I extends AsyncIface> extends
    ↪ org.apache.thrift.TBaseAsyncProcessor<I> {
    private static final Logger LOGGER = LoggerFactory.getLogger(
      ↪ AsyncProcessor.class.getName());
    public AsyncProcessor(I iface) {
      super(iface, getProcessMap(new HashMap<String, org.apache.
        ↪ thrift.AsyncProcessFunction<I, ? extends org.apache.
        ↪ thrift.TBase, ?>>()));
    }

    protected AsyncProcessor(I iface, Map<String,  org.apache.thrift.
      ↪ AsyncProcessFunction<I, ? extends  org.apache.thrift.TBase,
      ↪  ?>> processMap) {
      super(iface, getProcessMap(processMap));
    }

    private static <I extends AsyncIface> Map<String,  org.apache.
      ↪ thrift.AsyncProcessFunction<I, ? extends  org.apache.thrift
      ↪ .TBase,?>> getProcessMap(Map<String,  org.apache.thrift.
```

```
    ↪ AsyncProcessFunction<I, ? extends  org.apache.thrift.TBase,
    ↪  ?>> processMap) {
195   processMap.put("interpret", new interpret());
      return processMap;
    }

    public static class interpret<I extends AsyncIface> extends org.
    ↪ apache.thrift.AsyncProcessFunction<I, interpret_args,
    ↪ String> {
200   public interpret() {
        super("interpret");
      }

      public interpret_args getEmptyArgsInstance() {
205     return new interpret_args();
      }

      public AsyncMethodCallback<String> getResultHandler(
        final AsyncFrameBuffer fb, final int seqid) {
210     final org.apache.thrift.AsyncProcessFunction fcall = this;
        return new AsyncMethodCallback<String>() {
          public void onComplete(String o) {
            interpret_result result = new interpret_result();
            result.success = o;
215         try {
              fcall.sendResponse(
                fb,result, org.apache.thrift.protocol.TMessageType.
                ↪ REPLY,seqid);
              return;
            } catch (Exception e) {
220           LOGGER.error("Exception writing to internal frame
                ↪ buffer", e);
            }
            fb.close();
          }
          public void onError(Exception e) {
225         byte msgType = org.apache.thrift.protocol.TMessageType.
              ↪ REPLY;
            org.apache.thrift.TBase msg;
            interpret_result result = new interpret_result();
            {
              msgType = org.apache.thrift.protocol.TMessageType.
```

```
                    ↪ EXCEPTION;
230               msg = (org.apache.thrift.TBase)new org.apache.thrift.
                    ↪ TApplicationException(
                  org.apache.thrift.TApplicationException.
                    ↪ INTERNAL_ERROR, e.getMessage());
              }
              try {
                fcall.sendResponse(fb,msg,msgType,seqid);
235             return;
              } catch (Exception ex) {
                LOGGER.error("Exception writing to internal frame
                    ↪ buffer", ex);
              }
              fb.close();
240         }
          };
        }

        protected boolean isOneway() {
245       return false;
        }

        public void start(I iface, interpret_args args, org.apache.
            ↪ thrift.async.AsyncMethodCallback<String> resultHandler)
            ↪ throws TException {
          iface.interpret(args.code,resultHandler);
250     }
      }

    }

255 public static class interpret_args implements org.apache.thrift.
      ↪ TBase<interpret_args, interpret_args._Fields>, java.io.
      ↪ Serializable, Cloneable, Comparable<interpret_args>   {
      private static final org.apache.thrift.protocol.TStruct
        ↪ STRUCT_DESC = new org.apache.thrift.protocol.TStruct("
        ↪ interpret_args");

      private static final org.apache.thrift.protocol.TField
        ↪ CODE_FIELD_DESC = new org.apache.thrift.protocol.TField(
      "code", org.apache.thrift.protocol.TType.STRING, (short)1);
260
```

```
  private static final Map<Class<? extends IScheme>, SchemeFactory>
    ↪  schemes = new HashMap<Class<? extends IScheme>,
    ↪ SchemeFactory>();
  static {
    schemes.put(StandardScheme.class, new
      ↪ interpret_argsStandardSchemeFactory());
    schemes.put(TupleScheme.class, new
      ↪ interpret_argsTupleSchemeFactory());
  }

  public String code; // required

  /** The set of fields this struct contains, along with
    ↪ convenience methods
  for finding and manipulating them. */
  public enum _Fields implements org.apache.thrift.TFieldIdEnum {
    CODE((short)1, "code");

    private static final Map<String, _Fields> byName = new HashMap<
      ↪ String, _Fields>();

    static {
      for (_Fields field : EnumSet.allOf(_Fields.class)) {
        byName.put(field.getFieldName(), field);
      }
    }

    /**
     * Find the _Fields constant that matches fieldId, or null if
       ↪ its not found.
     */
    public static _Fields findByThriftId(int fieldId) {
      switch(fieldId) {
        case 1: // CODE
          return CODE;
        default:
          return null;
      }
    }

    /**
     * Find the _Fields constant that matches fieldId, throwing an
```

```
        ↪ exception
       * if it is not found.
       */
      public static _Fields findByThriftIdOrThrow(int fieldId) {
        _Fields fields = findByThriftId(fieldId);
300     if (fields == null) throw new IllegalArgumentException(
            "Field " + fieldId + " doesn't exist!");
        return fields;
      }

305   /**
       * Find the _Fields constant that matches name, or null if its
        ↪ not found.
       */
      public static _Fields findByName(String name) {
        return byName.get(name);
310   }

      private final short _thriftId;
      private final String _fieldName;

315   _Fields(short thriftId, String fieldName) {
        _thriftId = thriftId;
        _fieldName = fieldName;
      }

320   public short getThriftFieldId() {
        return _thriftId;
      }

      public String getFieldName() {
325     return _fieldName;
      }
    }

    // isset id assignments
330 public static final Map<_Fields, org.apache.thrift.meta_data.
      ↪ FieldMetaData> metaDataMap;
    static {
      Map<_Fields, org.apache.thrift.meta_data.FieldMetaData> tmpMap
        ↪ = new EnumMap<_Fields, org.apache.thrift.meta_data.
        ↪ FieldMetaData>(_Fields.class);
```

```
    tmpMap.put(_Fields.CODE, new org.apache.thrift.meta_data.
        ↪ FieldMetaData("code", org.apache.thrift.
        ↪ TFieldRequirementType.DEFAULT,
         new org.apache.thrift.meta_data.FieldValueMetaData(
           org.apache.thrift.protocol.TType.STRING)));
    metaDataMap = Collections.unmodifiableMap(tmpMap);
    org.apache.thrift.meta_data.FieldMetaData.addStructMetaDataMap(
      interpret_args.class, metaDataMap);
  }

  public interpret_args() {
  }

  public interpret_args(
    String code)
  {
    this();
    this.code = code;
  }

  /**
   * Performs a deep copy on <i>other</i>.
   */
  public interpret_args(interpret_args other) {
    if (other.isSetCode()) {
      this.code = other.code;
    }
  }

  public interpret_args deepCopy() {
    return new interpret_args(this);
  }

  @Override
  public void clear() {
    this.code = null;
  }

  public String getCode() {
    return this.code;
  }
```

```
  public interpret_args setCode(String code) {
    this.code = code;
    return this;
  }

  public void unsetCode() {
    this.code = null;
  }

  /** Returns true if field code is set (has been assigned a value)
  and false otherwise */
  public boolean isSetCode() {
    return this.code != null;
  }

  public void setCodeIsSet(boolean value) {
    if (!value) {
      this.code = null;
    }
  }

  public void setFieldValue(_Fields field, Object value) {
    switch (field) {
    case CODE:
      if (value == null) {
        unsetCode();
      } else {
        setCode((String)value);
      }
      break;

    }
  }

  public Object getFieldValue(_Fields field) {
    switch (field) {
    case CODE:
      return getCode();

    }
    throw new IllegalStateException();
  }
```

```
415
    /** Returns true if field corresponding to fieldID is set
    (has been assigned a value) and false otherwise */
    public boolean isSet(_Fields field) {
      if (field == null) {
420     throw new IllegalArgumentException();
      }

      switch (field) {
      case CODE:
425     return isSetCode();
      }
      throw new IllegalStateException();
    }

430 @Override
    public boolean equals(Object that) {
      if (that == null)
        return false;
      if (that instanceof interpret_args)
435     return this.equals((interpret_args)that);
      return false;
    }

    public boolean equals(interpret_args that) {
440   if (that == null)
        return false;

      boolean this_present_code = true && this.isSetCode();
      boolean that_present_code = true && that.isSetCode();
445   if (this_present_code || that_present_code) {
        if (!(this_present_code && that_present_code))
          return false;
        if (!this.code.equals(that.code))
          return false;
450   }

      return true;
    }

455 @Override
    public int hashCode() {
```

```
    List<Object> list = new ArrayList<Object>();

    boolean present_code = true && (isSetCode());
    list.add(present_code);
    if (present_code)
      list.add(code);

    return list.hashCode();
  }

  @Override
  public int compareTo(interpret_args other) {
    if (!getClass().equals(other.getClass())) {
      return getClass().getName().compareTo(other.getClass().
         ↪ getName());
    }

    int lastComparison = 0;

    lastComparison = Boolean.valueOf(isSetCode()).compareTo(other.
       ↪ isSetCode());
    if (lastComparison != 0) {
      return lastComparison;
    }
    if (isSetCode()) {
      lastComparison = org.apache.thrift.TBaseHelper.compareTo(this
         ↪ .code, other.code);
      if (lastComparison != 0) {
        return lastComparison;
      }
    }
    return 0;
  }

  public _Fields fieldForId(int fieldId) {
    return _Fields.findByThriftId(fieldId);
  }

  public void read(org.apache.thrift.protocol.TProtocol iprot)
     ↪ throws org.apache.thrift.TException {
    schemes.get(iprot.getScheme()).getScheme().read(iprot, this);
  }
```

```
495
    public void write(org.apache.thrift.protocol.TProtocol oprot)
       ↪ throws org.apache.thrift.TException {
      schemes.get(oprot.getScheme()).getScheme().write(oprot, this);
    }

500 @Override
    public String toString() {
      StringBuilder sb = new StringBuilder("interpret_args(");
      boolean first = true;

505   sb.append("code:");
      if (this.code == null) {
        sb.append("null");
      } else {
        sb.append(this.code);
510   }
      first = false;
      sb.append(")");
      return sb.toString();
    }
515
    public void validate() throws org.apache.thrift.TException {
      // check for required fields
      // check for sub-struct validity
    }
520
    private void writeObject(java.io.ObjectOutputStream out) throws
       ↪ java.io.IOException {
      try {
        write(new org.apache.thrift.protocol.TCompactProtocol(new org
           ↪ .apache.thrift.transport.TIOStreamTransport(out)));
      } catch (org.apache.thrift.TException te) {
525     throw new java.io.IOException(te);
      }
    }

    private void readObject(java.io.ObjectInputStream in) throws java
       ↪ .io.IOException, ClassNotFoundException {
530   try {
        read(new org.apache.thrift.protocol.TCompactProtocol(new org.
           ↪ apache.thrift.transport.TIOStreamTransport(in)));
```

```
      } catch (org.apache.thrift.TException te) {
        throw new java.io.IOException(te);
      }
    }

    private static class interpret_argsStandardSchemeFactory
       ↪ implements SchemeFactory {
      public interpret_argsStandardScheme getScheme() {
        return new interpret_argsStandardScheme();
      }
    }

    private static class interpret_argsStandardScheme extends
       ↪ StandardScheme<interpret_args> {

      public void read(
        org.apache.thrift.protocol.TProtocol iprot, interpret_args
           ↪ struct) throws org.apache.thrift.TException {
        org.apache.thrift.protocol.TField schemeField;
        iprot.readStructBegin();
        while (true)
        {
          schemeField = iprot.readFieldBegin();
          if (schemeField.type == org.apache.thrift.protocol.TType.
             ↪ STOP) {
            break;
          }
          switch (schemeField.id) {
            case 1: // CODE
              if (schemeField.type == org.apache.thrift.protocol.
                 ↪ TType.STRING) {
                struct.code = iprot.readString();
                struct.setCodeIsSet(true);
              } else {
                org.apache.thrift.protocol.TProtocolUtil.skip(iprot,
                   ↪ schemeField.type);
              }
              break;
            default:
              org.apache.thrift.protocol.TProtocolUtil.skip(iprot,
                 ↪ schemeField.type);
          }
```

```
        iprot.readFieldEnd();
      }
      iprot.readStructEnd();

      // check for required fields of primitive type, which can't
         ↪ be checked in the validate method
      struct.validate();
    }

    public void write(
      org.apache.thrift.protocol.TProtocol oprot, interpret_args
         ↪ struct) throws org.apache.thrift.TException {
      struct.validate();

      oprot.writeStructBegin(STRUCT_DESC);
      if (struct.code != null) {
        oprot.writeFieldBegin(CODE_FIELD_DESC);
        oprot.writeString(struct.code);
        oprot.writeFieldEnd();
      }
      oprot.writeFieldStop();
      oprot.writeStructEnd();
    }

  }

  private static class interpret_argsTupleSchemeFactory implements
     ↪ SchemeFactory {
    public interpret_argsTupleScheme getScheme() {
      return new interpret_argsTupleScheme();
    }
  }

  private static class interpret_argsTupleScheme extends
     ↪ TupleScheme<interpret_args> {

    @Override
    public void write(
      org.apache.thrift.protocol.TProtocol prot, interpret_args
         ↪ struct) throws org.apache.thrift.TException {
      TTupleProtocol oprot = (TTupleProtocol) prot;
      BitSet optionals = new BitSet();
```

```
        if (struct.isSetCode()) {
605       optionals.set(0);
        }
        oprot.writeBitSet(optionals, 1);
        if (struct.isSetCode()) {
          oprot.writeString(struct.code);
610     }
      }

      @Override
      public void read(
615     org.apache.thrift.protocol.TProtocol prot, interpret_args
          ↪ struct) throws org.apache.thrift.TException {
        TTupleProtocol iprot = (TTupleProtocol) prot;
        BitSet incoming = iprot.readBitSet(1);
        if (incoming.get(0)) {
          struct.code = iprot.readString();
620       struct.setCodeIsSet(true);
        }
      }
    }

625 }

    public static class interpret_result implements org.apache.thrift.
       ↪ TBase<interpret_result, interpret_result._Fields>, java.io.
       ↪ Serializable, Cloneable, Comparable<interpret_result>   {
      private static final org.apache.thrift.protocol.TStruct
         ↪ STRUCT_DESC = new org.apache.thrift.protocol.TStruct("
         ↪ interpret_result");

630   private static final org.apache.thrift.protocol.TField
         ↪ SUCCESS_FIELD_DESC = new org.apache.thrift.protocol.TField
         ↪ ("success", org.apache.thrift.protocol.TType.STRING, (short
         ↪ )0);

      private static final Map<Class<? extends IScheme>, SchemeFactory>
         ↪  schemes = new HashMap<Class<? extends IScheme>,
         ↪ SchemeFactory>();
      static {
        schemes.put(StandardScheme.class, new
           ↪ interpret_resultStandardSchemeFactory());
```

```
635     schemes.put(TupleScheme.class, new
           ↪ interpret_resultTupleSchemeFactory());
    }

    public String success; // required

640 /** The set of fields this struct contains, along with
       ↪ convenience methods
    for finding and manipulating them. */
    public enum _Fields implements org.apache.thrift.TFieldIdEnum {
      SUCCESS((short)0, "success");

645   private static final Map<String, _Fields> byName = new HashMap<
         ↪ String, _Fields>();

      static {
        for (_Fields field : EnumSet.allOf(_Fields.class)) {
          byName.put(field.getFieldName(), field);
650     }
      }

      /**
       * Find the _Fields constant that matches fieldId, or null if
          ↪ its not found.
655    */
      public static _Fields findByThriftId(int fieldId) {
        switch(fieldId) {
          case 0: // SUCCESS
            return SUCCESS;
660       default:
            return null;
        }
      }

665   /**
       * Find the _Fields constant that matches fieldId, throwing an
          ↪ exception
       * if it is not found.
       */
      public static _Fields findByThriftIdOrThrow(int fieldId) {
670     _Fields fields = findByThriftId(fieldId);
        if (fields == null) throw new IllegalArgumentException(
```

```
          "Field " + fieldId + " doesn't exist!");
      return fields;
    }

    /**
     * Find the _Fields constant that matches name, or null if its
       ↪ not found.
     */
    public static _Fields findByName(String name) {
      return byName.get(name);
    }

    private final short _thriftId;
    private final String _fieldName;

    _Fields(short thriftId, String fieldName) {
      _thriftId = thriftId;
      _fieldName = fieldName;
    }

    public short getThriftFieldId() {
      return _thriftId;
    }

    public String getFieldName() {
      return _fieldName;
    }
  }

  // isset id assignments
  public static final Map<_Fields, org.apache.thrift.meta_data.
     ↪ FieldMetaData> metaDataMap;
  static {
    Map<_Fields, org.apache.thrift.meta_data.FieldMetaData> tmpMap
       ↪ = new EnumMap<_Fields, org.apache.thrift.meta_data.
       ↪ FieldMetaData>(_Fields.class);
    tmpMap.put(_Fields.SUCCESS, new org.apache.thrift.meta_data.
       ↪ FieldMetaData("success", org.apache.thrift.
       ↪ TFieldRequirementType.DEFAULT,
        new org.apache.thrift.meta_data.FieldValueMetaData(
          org.apache.thrift.protocol.TType.STRING)));
    metaDataMap = Collections.unmodifiableMap(tmpMap);
```

```
      org.apache.thrift.meta_data.FieldMetaData.addStructMetaDataMap(
        interpret_result.class, metaDataMap);
710   }

    public interpret_result() {
    }

715 public interpret_result(
      String success)
    {
      this();
      this.success = success;
720 }

    /**
     * Performs a deep copy on <i>other</i>.
     */
725 public interpret_result(interpret_result other) {
      if (other.isSetSuccess()) {
        this.success = other.success;
      }
    }
730
    public interpret_result deepCopy() {
      return new interpret_result(this);
    }

735 @Override
    public void clear() {
      this.success = null;
    }

740 public String getSuccess() {
      return this.success;
    }

    public interpret_result setSuccess(String success) {
745   this.success = success;
      return this;
    }

    public void unsetSuccess() {
```

```
750       this.success = null;
        }

        /** Returns true if field success is set (has been assigned a
           ↪ value) and false otherwise */
        public boolean isSetSuccess() {
755       return this.success != null;
        }

        public void setSuccessIsSet(boolean value) {
          if (!value) {
760         this.success = null;
          }
        }

        public void setFieldValue(_Fields field, Object value) {
765       switch (field) {
          case SUCCESS:
            if (value == null) {
              unsetSuccess();
            } else {
770           setSuccess((String)value);
            }
            break;

          }
775     }

        public Object getFieldValue(_Fields field) {
          switch (field) {
          case SUCCESS:
780         return getSuccess();

          }
          throw new IllegalStateException();
        }
785
        /** Returns true if field corresponding to fieldID is set (has
           ↪ been assigned a value) and false otherwise */
        public boolean isSet(_Fields field) {
          if (field == null) {
            throw new IllegalArgumentException();
```

```
790       }

          switch (field) {
          case SUCCESS:
            return isSetSuccess();
795       }
          throw new IllegalStateException();
        }

        @Override
800     public boolean equals(Object that) {
          if (that == null)
            return false;
          if (that instanceof interpret_result)
            return this.equals((interpret_result)that);
805       return false;
        }

        public boolean equals(interpret_result that) {
          if (that == null)
810         return false;

          boolean this_present_success = true && this.isSetSuccess();
          boolean that_present_success = true && that.isSetSuccess();
          if (this_present_success || that_present_success) {
815         if (!(this_present_success && that_present_success))
              return false;
            if (!this.success.equals(that.success))
              return false;
          }
820
          return true;
        }

        @Override
825     public int hashCode() {
          List<Object> list = new ArrayList<Object>();

          boolean present_success = true && (isSetSuccess());
          list.add(present_success);
830       if (present_success)
            list.add(success);
```

```
      return list.hashCode();
    }

    @Override
    public int compareTo(interpret_result other) {
      if (!getClass().equals(other.getClass())) {
        return getClass().getName().compareTo(other.getClass().
            ↪ getName());
      }

      int lastComparison = 0;

      lastComparison = Boolean.valueOf(isSetSuccess()).compareTo(
          ↪ other.isSetSuccess());
      if (lastComparison != 0) {
        return lastComparison;
      }
      if (isSetSuccess()) {
        lastComparison = org.apache.thrift.TBaseHelper.compareTo(this
            ↪ .success, other.success);
        if (lastComparison != 0) {
          return lastComparison;
        }
      }
      return 0;
    }

    public _Fields fieldForId(int fieldId) {
      return _Fields.findByThriftId(fieldId);
    }

    public void read(org.apache.thrift.protocol.TProtocol iprot)
        ↪ throws org.apache.thrift.TException {
      schemes.get(iprot.getScheme()).getScheme().read(iprot, this);
    }

    public void write(org.apache.thrift.protocol.TProtocol oprot)
        ↪ throws org.apache.thrift.TException {
      schemes.get(oprot.getScheme()).getScheme().write(oprot, this);
      }
```

```
    @Override
870 public String toString() {
      StringBuilder sb = new StringBuilder("interpret_result(");
      boolean first = true;

      sb.append("success:");
875   if (this.success == null) {
        sb.append("null");
      } else {
        sb.append(this.success);
      }
880   first = false;
      sb.append(")");
      return sb.toString();
    }

885 public void validate() throws org.apache.thrift.TException {
      // check for required fields
      // check for sub-struct validity
    }

890 private void writeObject(java.io.ObjectOutputStream out) throws
       ↪ java.io.IOException {
      try {
        write(new org.apache.thrift.protocol.TCompactProtocol(new org
           ↪ .apache.thrift.transport.TIOStreamTransport(out)));
      } catch (org.apache.thrift.TException te) {
        throw new java.io.IOException(te);
895   }
    }

    private void readObject(java.io.ObjectInputStream in) throws java
       ↪ .io.IOException, ClassNotFoundException {
      try {
900     read(new org.apache.thrift.protocol.TCompactProtocol(new org.
           ↪ apache.thrift.transport.TIOStreamTransport(in)));
      } catch (org.apache.thrift.TException te) {
        throw new java.io.IOException(te);
      }
    }
905

    private static class interpret_resultStandardSchemeFactory
```

```
    ↪ implements SchemeFactory {
  public interpret_resultStandardScheme getScheme() {
    return new interpret_resultStandardScheme();
  }
}

private static class interpret_resultStandardScheme extends
    ↪ StandardScheme<interpret_result> {

  public void read(org.apache.thrift.protocol.TProtocol iprot,
      ↪ interpret_result struct) throws org.apache.thrift.
      ↪ TException {
    org.apache.thrift.protocol.TField schemeField;
    iprot.readStructBegin();
    while (true)
    {
      schemeField = iprot.readFieldBegin();
      if (schemeField.type == org.apache.thrift.protocol.TType.
          ↪ STOP) {
        break;
      }
      switch (schemeField.id) {
        case 0: // SUCCESS
          if (schemeField.type == org.apache.thrift.protocol.
              ↪ TType.STRING) {
            struct.success = iprot.readString();
            struct.setSuccessIsSet(true);
          } else {
            org.apache.thrift.protocol.TProtocolUtil.skip(iprot,
                ↪ schemeField.type);
          }
          break;
        default:
          org.apache.thrift.protocol.TProtocolUtil.skip(iprot,
              ↪ schemeField.type);
      }
      iprot.readFieldEnd();
    }
    iprot.readStructEnd();

    // check for required fields of primitive type, which can't
        ↪ be checked in the validate method
```

```
940         struct.validate();
          }

          public void write(org.apache.thrift.protocol.TProtocol oprot,
             ↪ interpret_result struct) throws org.apache.thrift.
             ↪ TException {
            struct.validate();
945
            oprot.writeStructBegin(STRUCT_DESC);
            if (struct.success != null) {
              oprot.writeFieldBegin(SUCCESS_FIELD_DESC);
              oprot.writeString(struct.success);
950           oprot.writeFieldEnd();
            }
            oprot.writeFieldStop();
            oprot.writeStructEnd();
          }
955
        }

        private static class interpret_resultTupleSchemeFactory
           ↪ implements SchemeFactory {
          public interpret_resultTupleScheme getScheme() {
960         return new interpret_resultTupleScheme();
          }
        }

        private static class interpret_resultTupleScheme extends
           ↪ TupleScheme<interpret_result> {
965
          @Override
          public void write(org.apache.thrift.protocol.TProtocol prot,
             ↪ interpret_result struct) throws org.apache.thrift.
             ↪ TException {
            TTupleProtocol oprot = (TTupleProtocol) prot;
            BitSet optionals = new BitSet();
970         if (struct.isSetSuccess()) {
              optionals.set(0);
            }
            oprot.writeBitSet(optionals, 1);
            if (struct.isSetSuccess()) {
975           oprot.writeString(struct.success);
```

```
        }
      }

      @Override
      public void read(
        org.apache.thrift.protocol.TProtocol prot, interpret_result
          ↪ struct) throws org.apache.thrift.TException {
        TTupleProtocol iprot = (TTupleProtocol) prot;
        BitSet incoming = iprot.readBitSet(1);
        if (incoming.get(0)) {
          struct.success = iprot.readString();
          struct.setSuccessIsSet(true);
        }
      }
    }

  }

}
```