UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
DEPARTAMENTO DE INFORMÁTICA
MESTRADO EM INFORMÁTICA

VICTOR AMORIM DOS SANTOS

UMA ABORDAGEM BASEADA EM PADRÕES PARA CONSTRUÇÃO DE MODELOS CONCEITUAIS EM ONTOUML

VITÓRIA – ES, Brasil

OUTUBRO 2015

VICTOR AMORIM DOS SANTOS

UMA ABORDAGEM BASEADA EM PADRÕES PARA CONSTRUÇÃO DE MODELOS CONCEITUAIS EM ONTOUML

Dissertação submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Mestre em Informática.

VITÓRIA – ES, Brasil

OUTUBRO 2015

VICTOR AMORIM DOS SANTOS

UMA ABORDAGEM BASEADA EM PADRÕES PARA CONSTRUÇÃO DE MODELOS CONCEITUAIS EM ONTOUML

Dissertação submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Mestre em Informática.

Aprovada em 27 de Outubro de 2015.

COMISSÃO EXAMINADORA

_____
Prof. Dr. Giancarlo Guizzardi
Universidade Federal do Espírito Santo (UFES) (Orientador)

_____
Prof. Dr. Ricardo de Almeida Falbo,
Universidade Federal do Espírito Santo (UFES)

_____
Prof. Dr. Maxwell Eduardo Monteiro
Instituto Federal do Espírito Santo – SERRA (IFES)

_____
Prof. Dr. Mateus Conrad Barcellos da Costa
Instituto Federal do Espírito Santo – SERRA (IFES)

VITÓRIA, OUTUBRO 2015

DEDICATÓRIA

Para aqueles que tornaram possível essa dissertação.

Nobody said it was easy
It's such a shame for us to part
Nobody said it was easy
No one ever said it would be this hard – **Coldplay**

Don't worry about a thing,
'Cause every little thing gonna be all right! – **Bob Marley**

AGRADECIMENTO

Primeiramente, gostaria de agradecer a meus familiares por terem acreditado e me apoiado ao longo desses anos, em especial a meus pais Juvenal e Mariene e a meu irmão Angelo Thiago. Em especial, a Luanna, por ter me acompanhado por todo esse tempo, com palavras e gestos que fizeram toda a diferença.

Também gostaria de agradecer aos que contribuíram diretamente para o sucesso desse trabalho, em especial: Giancarlo Guizzardi, Maxwell, Fabiano, Cássio, Jean-Rémi, Archimedes e Pedro Paulo. Aos colegas de laboratório por terem contribuído com profundas discussões sobre os mais diversos temas, em especial: Freddy, John, Laylla, Missael, Bernardo, Bassetti, Gabriel Miranda, Carlos, Gabriel Ramaldes, Juan Calles, Sobral. Enfim, gostaria de agradecer a todos que tiveram parte durante o desenvolvimento deste trabalho e dizer que sem eles, provavelmente, isto não seria possível.

Por fim, acredito que mais importante do que os resultados presentes aqui, foi o processo pessoal de se tornar um pesquisador, de aprender a questionar e debater sobre as verdades, que fez diferença na minha vida como pessoa e que levarei comigo.

RESUMO

Recentemente tem crescido o interesse pelo uso de linguagens conceituais bem fundamentadas como suporte a fase de analise de domínio na Engenharia de Software. Nesse contexto, o presente trabalho surge como mais uma iniciativa de se utilizar as meta-propriedades da linguagem OntoUML, derivada da ontologia de fundamentação UFO (Unified Foundational Ontology), as quais governam a forma como o processo de modelagem funciona, para propor uma abordagem baseada em padrões para construção de modelos conceituais bem fundamentados utilizando os padrões da linguagem OntoUML.

Neste trabalho apresentamos um Catálogo de Padrões para linguagem OntoUML visando prover aos usuários mais avançadas da linguagem uma ferramenta de aumento de produtividade e de melhoria na acurácia dos modelos. Além disso, uma vez que o uso de linguagens ontológicas tendem a aumentar a complexidade do processo de modelagem, propomos também uma serie de procedimentos para auxiliar na construção dos modelos OntoUML. Por fim, unirmos estas duas estratégias para criar uma abordagem orientada a eventos de modelagem que provê ao usuário uma lista de padrões a serem usados para cada adição de uma nova classe ao modelo.

Palavras-chave: Modelagem Conceitual, OntoUML, UFO, Foundational Ontology Patterns, Ontologia.

ABSTRACT

In recent years, there has been a growing interest for well-founded conceptual modeling languages in support to domain analysis in Software Engineering. In this context, this work is a contribution to ontology-driven conceptual modeling, in general, and to the OntoUML language, in particular.

The work contributes to this approach by proposing a catalog of Ontology Design Patterns for OntoUML, and a set of procedures for supporting the pattern-based construction of OntoUML models using the foundational patterns in this catalog. Finally, the work implements in a computational tool the complete mechanism that allows building OntoUML models by applying these design patterns following the proposed procedures.

Key-Word: Conceptual Modeling, OntoUML, UFO, Foundational Ontology Patterns, Ontology.

LISTA DE FIGURAS

Figura 1 - Fragmento do Metamodelo de UFO – Adaptado de (GUIZZARDI, 2005) 23
Figura 2 - Exemplo de instâncias rígidas e anti-rígidas 25
Figura 3 - Exemplo de Mixin agrupando tipos de identidade diferentes 28
Figura 4 - Exemplo de um modelo de casamento 30
Figura 5 - Modelo de casamento em OntoUML 31
Figura 6 - Fragmento de UFO - Extraído de (ZAMBORLINI, 2011) 31
Figura 7 – Fragmento de UFO sobre as relações Meronymics (ZAMBORLINI, 2011) 32
Figura 8 – Fragmento do metamodelo de OntoUML (GUIZZARDI, 2005) 34
Figura 9 - Visão geral das funcionalidades do OLED 36
Figura 10 - Exemplo de diagrama feito no OLED (GUERSON et al., 2015) 36
Figura 11 - Relator FOP 41
Figura 12 - Role FOP 41
Figura 13 - Role-Relator FOP 42
Figura 14 - Principle of Identity FOPs 44
Figura 15 - Exemplo de uso: Principle of Identity FOP 45
Figura 16 - Simple Relator FOP 45
Figura 17 - Multiple Generic Relator FOP 46
Figura 18 - Generic Relator FOP 46
Figura 19 - General Relator FOP 47
Figura 20 - Exemplo de Uso - General Relator FOP 47
Figura 21 - RoleMixin Dependence FOP 48
Figura 22 - Exemplo de Uso - RoleMixin Dependence FOP 49
Figura 23 - Role Partition FOP 50
Figura 24 - Phase Partition FOP 50
Figura 25 - Subkind Partition FOP 50
Figura 26 - Category Partition FOP 51
Figura 27 - RoleMixin Partition FOP 51
Figura 28 - Mode Partition FOP 51
Figura 29 - Relator Partition FOP 52
Figura 30 - Exemplo de Uso - Subkind Partition FOP 52
Figura 31 - Exemplo de Uso - Phase Partition FOP 53
Figura 32 - Category Generalization FOP 54
Figura 33 - Mixin Generalization FOP 54
Figura 34 - RoleMixin Generalization FOP 55
Figura 35 - Exemplo de Uso - Mixin Generalization FOP 55
Figura 36 - Formal FOP 56
Figura 37 - Characterization FOP 56
Figura 38 - Exemplo de Uso - Characterization FOP 57
Figura 39 - Generic Meronymic Form 57
Figura 40 - Exemplo de Uso - Component Of FOP 59
Figura 41 - Evento de Modelagem - Novo Elemento 62
Figura 42 - Classificação geral dos procedimentos 64
Figura 43 - Procedimento: Substance Sortal 65
Figura 44 - Procedimento: Subkind 66
Figura 45 - Procedimento: Role 67

Figura 46 - Procedimento: Phase 68
Figura 47 - Procedimento: Relator 69
Figura 48 - Definição das cardinalidades para relacionamentos com Relators (GUIZZARDI, 2005) 70
Figura 49 - Procedimento: RoleMixin 71
Figura 50 - Procedimento: Category 72
Figura 51 - Procedimento: Mixin 73
Figura 52 - Procedimento: Mode 75
Figura 53 - Procedimento: Partition 77
Figura 54 - Procedimento: Generalization 79
Figura 55 - Procedimento: Relações de Dependência 81
Figura 56 - Procedimento: Derivation 82
Figura 57 - Procedimento: Relações de Todo-Parte 85
Figura 58 - Biblioteca de Padrões 88
Figura 59 - Janela de Configuração de Padrão 88
Figura 60 - Janela de Ajuda 90
Figura 61 - Exemplo - Parte I 91
Figura 62 - Exemplo: Parte II 92
Figura 63 - Exemplo: Parte III 92
Figura 64 - Exemplo: Parte IV 93
Figura 65 - Exemplo: Parte V 94
Figura 66 - Exemplo: Parte VI 95
Figura 67 - Algoritmo de Verificação das Restrições em Modelos OntoUML 96
Figura 68 - Tela de Verificação do Modelo 97
Figura 69 - Aplicando o Padrão Multiple Generic Relator 98

LISTA DE TABELAS

Tabela 1 - Possibilidades de Partições 76
Tabela 2 - Possibilidades de Generalização 78
Tabela 3 – Possibilidades das Relações de Dependência 80
Tabela 4 - Possibilidades das Relações Todo-Parte 83

SUMÁRIO

CAPÍTULO 1. Introdução 14

1.1 Introdução 14

1.2 Contexto 16

1.3 Motivação 18

1.4 Objetivos da Pesquisa 19

1.5 Método de Pesquisa 19

1.6 Organização da Dissertação 20

CAPÍTULO 2. Ontologias e Padrões Ontológicos 21

2.1 Ontologias 21

2.2 A Ontologia de Fundamentação Unificada 22

2.3 OntoUML 33

2.4 OLED: Um Editor OntoUML 35

2.5 Considerações Finais do Capítulo 38

CAPÍTULO 3. Padrões OntoUML 40

3.1 Extração de Padrões Ontológicos de Fundamentação a partir de UFO-A 40

3.2 O Catálogo de Padrões OntoUML 43

3.3 Trabalhos Relacionados 59

3.4 Considerações Finais do Capítulo 60

CAPÍTULO 4. Procedimento para Construção de Modelos OntoUML 61

4.1 Um Processo Geral para a Construção de Modelos OntoUML 61

4.2 Procedimentos Básicos 64

4.2.1 Substance Sortal 64

4.2.2 Subkind 65

4.2.3 Role 66

4.2.4 Phase 67

4.2.5 Relator 69

4.2.6 RoleMixin 70

4.2.7 Category 71

4.2.8 Mixin 72

4.2.9 Mode 74

4.3 Procedimentos Relacionais 75

4.3.1 Procedimento de Partição 76

4.3.2 Procedimento de Generalização 78

4.3.3 Procedimento de Relação 79

CAPÍTULO 5. Avaliação da Abordagem 87

5.1 Apoio ao Uso de Padrões OntoUML em OLED 87

5.2 Exemplo de Aplicação: Universidade 90

5.3 Algoritmo de Verificação do Atendimento de Restrições em Modelos OntoUML 95

5.4 Trabalhos Relacionados 98

CAPÍTULO 6. Considerações Finais e Perspectivas Futuras 100

6.1 Considerações Finais 100

6.2 Perspectivas Futuras 101

Referências 103

CAPÍTULO 1. INTRODUÇÃO

1.1 INTRODUÇÃO

Informação é importante para todo tipo de tomada de decisão. Sem o entendimento correto do problema a ser resolvido, indivíduos, organizações, comunidades e governos podem não conseguir chegar a uma conclusão sobre o problema ou mesmo analisar os efeitos de suas ações (GUIZZARDI, 2014). Neste sentido, diversas técnicas e métodos surgem para auxiliar na estruturação e compartilhamento da informação. Dentre essas, temos a área de Modelagem Conceitual a qual tem por finalidade auxiliar na modelagem do conhecimento, na formulação do entendimento e na simulação dos conceitos relacionados a um determinado domínio do conhecimento através da representação de Modelos Conceituais. Em outras palavras, a disciplina de modelagem conceitual pode ser vista como a atividade de descrição dos aspectos físicos e sociais do mundo com o propósito de formalizar o entendimento e a comunicação entre seres humanos (MYLOPOULOS, 1992).

Como apontado por (MYLOPOULOS, 1992), a intenção de um modelo conceitual é ser usado e entendido por seres humanos, não máquinas, tornando seu uso prático e objetivo. Assim, a representação do conhecimento de forma estruturada e compartilhada tem sido usada por várias áreas de estudo. Destacamos neste contexto, as técnicas para representação do conhecimento na Engenharia de Software para construção de software, as quais tem-se mostrado promissora (JONES, 2010). O uso de modelos conceituais originados nas fases de levantamento de requisito, análise e projeto demonstram a importância e os benefícios destas técnicas para formalização e compartilhamento do conhecimento. Nesses modelos, temos representado as informações pertinentes sobre o domínio para cada uma das fases do processo de desenvolvimento de software. Desta forma, possibilitando a troca de informação entre todos os membros da equipe.

Entretanto, embora as técnicas de representação de conhecimento, como a modelagem conceitual, sirvam para a troca de informação entre seres humanos, a não formalização deste processo de troca de informação pode resultar em ambiguidades e mal-entendimento entre as

partes envolvidas, resultado em um processo de comunicação ineficiente. Em outras palavras, sem uma padronização dos elementos usados para representar a informação (e.g. conceitos, diagramas, elementos diagramáticos), o resultado pode ser uma visão turva do real sentido da informação. Como exemplo, temos o problema típico da troca de informação entre especialistas de áreas distintas, uma vez que cada um possuí uma expertise própria, com seus jargões e conceitos específicos, a troca de informações através de modelos conceituais tradicionais tende a se tornar ineficiente.

Ainda que auxiliando no processo de troca de informação, a modelagem conceitual tradicional (semanticamente ineficiente) apresenta algumas deficiências quanto a capacidade de preservar o sentido inicial da informação, uma vez que não define um formato padrão de representação dos elementos do modelo. Neste sentido, novas técnicas de enriquecimento semântico surgem como iniciativas para preservação da informação. A modelagem conceitual bem-fundamentada surge como uma solução para este problema. Baseada em linguagens que formalizam tanto a representação da informação quanto ao compartilhamento de informação, estas técnicas proveem formas mais rigorosas de se modelar. Utilizando resultados oriundos de teorias filosóficas e cognitivas estas abordagens possuem constructos para modelagem (e.g. tipos específicos de elementos de classificação) que possibilitam expressar de maneira formal todos os diferentes tipos de conceitos do domínio, diminuindo as incertezas quanto a sua interpretação futura. Desta forma, além de prover um arcabouço de construtos para modelagem independente de domínio também propiciam uma correta definição do modelo.

Dentre as linguagens existentes para modelagem conceitual destacamos a Web Ontology Language (OWL) (PATEL-SCHNEIDER; HAYES; HORROCKS, 2004) e OntoUML (GUIZZARDI, 2005). Enquanto OWL surge como um aprimoramento das técnicas para modelagem e compartilhamento de informação na web, OntoUML é uma extensão de UML para construção de modelos conceituais bem-fundamentados. Embora OWL também seja usado como linguagem para construção de modelos conceituais, sua sintaxe é baseada em XML o que dificulta seu uso por seres humanos. Já OntoUML possui sua sintaxe concreta (sintaxe visual) baseada nos diagramas de classe de UML e se baseia nos conceitos propostos pela Unified Foundational Ontology (UFO) (GUIZZARDI, 2005) (teoria sobre as principais classes de objetos usados na modelagem conceitual) para formalizar o processo de modelagem conceitual. Como veremos no decorrer desta dissertação, tanto UFO quanto

OntoUML provem uma variedade de conceitos para modelagem conceitual, os quais possuem objetivos e formas de aplicação específicos.

Uma vez que as formas de uso dos conceitos de OntoUML são estruturados segundo as regras de fundamentação de UFO, essas regras regem *as formas* como os tipos de OntoUML podem ser usados (seus relacionamentos, dependências, cardinalidades) na construção de modelos específicos. Seguindo a aplicação dessas regras, as *formas* de se criar modelos usando a linguagem se apresentam como blocos de uso que se repetem no modelo, caracterizando *padrões da linguagem* (RUY et al., 2014). Como veremos a seguir, estes blocos de padrões da linguagem possibilitam a construção dos modelos de forma estruturada e incremental. Tendo a analise da formação e identificação destes padrões como ponto de partida, esta dissertação visa analisar como estes blocos podem ser extraídos em forma de padrões recorrentes da linguagem visando cataloga-los em um catálogo de padrões da linguagem, bem como possibilitar o reuso destes padrões durante o processo de construção de modelos.

1.2 CONTEXTO

O estudo do ser enquanto ser (*being qua being*) possuí diferentes sentidos nas diversas áreas da ciência (GUARINO; OBERLE; STAAB, 2009). Esta área de estudo da Filosofia é chamada de Ontologia e como essência prescreve a busca por um entendimento da realidade. O estudo sobre Ontologia frequentemente lida com questões sobre as entidades que existem (ou deveriam existir) e como estas entidades poderiam ser agrupadas e relacionadas entre si por uma hierarquia ou divisão em grupos (de acordo com suas similaridades ou diferenças).

Embora Ontologia seja uma área de estudo da Filosofia, algumas iniciativas da área de Computação buscaram introduzir as teorias dessa área de conhecimento à modelagem conceitual baseada em Ontologias. Neste cenário, OntoUML é introduzida como uma linguagem ontológica de alta expressividade para modelagem conceitual bem-fundamentada. Fundamentada nos aspectos filosóficos e cognitivas discutidos na Unified Foundational Ontology (UFO) (GUIZZARDI, 2005), OntoUML tem sido amplamente aplicada com sucesso em projetos de diferentes domínios (GUIZZARDI et al., 2015). Além disso, as distinções ontológicas impostas pela linguagem tem sido discutida em diversos trabalhos (GUIZZARDI, 2014). Entretanto, apesar da crescente popularidade da linguagem, OntoUML

tem sido considerada complexa para modeladores novatos (GUIZZARDI; GRAÇAS; GUIZZARDI, 2011). Consequentemente, diversos trabalhos propiciam diferentes técnicas para assistir estes usuários durante o processo de modelagem (GUIZZARDI, 2014).

Como supracitado, diversos trabalhos têm tomado proveito das distinções ontológicas (regras de fundamentação da linguagem) que governam a construção dos modelos OntoUML para auxiliar novos modeladores. Essas regras controlam o uso da linguagem, bem como as possíveis combinações de seus elementos. Na construção de um modelo em OntoUML o usuário é induzido a construir modelos via a combinação de padrões pré-existentes da linguagem embutidos nas regras de fundamentação. Dessa forma temos que a cada adição de um novo elemento da linguagem ao modelo, uma sequência de passos (recorrentes) precisa ser executada para manter o modelo alinhado com as regras de fundamentação de UFO (GUIZZARDI; GRAÇAS; GUIZZARDI, 2011).

Colocando isso de uma forma diferente, em contraste com linguagens ontologicamente neutras, como UML (Unified Modeling Language) e EER (Enhanced Entity-Relationship) temos que OntoUML apresenta-se como uma *Pattern Language* (Linguagem de Padrão) (RUY et al., 2014). Dessa forma, a sequência de passos mencionado anteriormente representam padrões que constituem primitivas de modelagem com um alto grau de abstração, quando comparados com as primitivas mais comuns da modelagem conceitual, como: classes, associações, especializações, dentre outras. Além disso, esse alto nível de abstração somente permite que os elementos de modelagem sejam combinados entre si de uma maneira restritiva, caracterizando um modelo formado por blocos de padrões recorrentes (GUIZZARDI; GRAÇAS; GUIZZARDI, 2011).

Baseado nessas propriedades, trabalhos como (GUIZZARDI; GRAÇAS; GUIZZARDI, 2011) apresentam uma abordagem focada na analise destes blocos de padrões de modelagem de OntoUML e de suas regras de fundamentação. Desses blocos de padrões, os autores extraíram uma sequência de passos a serem seguidos visando a construção do modelo utilizando estes padrões recorrentes como base. Estes procedimentos garantem a construção correta dos modelos e o alinhamento do modelo com as regras de fundamentação da linguagem. Entretanto, o trabalho supracitado bem como a dissertação intitulada “Suporte Automatizado para Construção de Modelos Conceituais bem Fundamentados” (GRAÇAS, 2010) associada a ele apenas lidam com uma pequena porção dos tipos de OntoUML

deixando em aberto uma análise completa dos outros tipos da linguagem. Dessa forma, o presente trabalho pretende dar continuidade a esta analise buscando abranger todos os tipos de OntoUML, identificando os padrões existentes para estes tipos, bem como a formalização dos procedimentos de criação de cada um destes elementos.

1.3 MOTIVAÇÃO

Com já mencionado, o trabalho apresentado em (GRAÇAS, 2010) é uma demonstração da capacidade dos modelos OntoUML serem vistos como um arranjo de blocos de padrões. Outro forte indicador desta capacidade é a discussão apresentada em (GUIZZARDI, 2005), onde Guizzardi apresenta os primeiros padrões identificados para OntoUML e toda a motivação por trás deles. Dessa forma, esse trabalho busca investigar de forma mais profunda essa propriedade de OntoUML buscando dar continuidade ao trabalho de Graças, além de contribuir com novos horizontes de estudo para modelagem conceitual bem-fundamentada.

Assim como para seus tipos, os padrões de OntoUML também apresentam interdependências. Como mostrado em (GUIZZARDI; GRAÇAS; GUIZZARDI, 2011) e (GUIZZARDI, 2005), essa dependência dos padrões da linguagem se manifestam em uma cadeia de aplicação de padrões. Todavia, a análise apresentada em (GUIZZARDI; GRAÇAS; GUIZZARDI, 2011) se restringe a um pequeno grupo de OntoUML (i.e. Kind, Subkind, Role e Phase – conceitos discutidos com mais detalhes nas próximas seções) deixando de lado uma grande parte dos tipos da linguagem. Já os padrões apresentado em (GUIZZARDI, 2005) são apenas identificados por suas recorrências em modelos, não sendo discutido seus procedimentos de formação e suas variações de aplicação.

Diante do exposto, o presente trabalho pretende investigar com mais detalhes toda a gama de tipos existentes em OntoUML. Partindo dos padrões identificados em (GUIZZARDI, 2005) e dos padrões e procedimentos apresentados em (GUIZZARDI; GRAÇAS; GUIZZARDI, 2011), pretende-se formalizar em um Catálogo de Padrões de OntoUML todos os padrões gerados pelas regras de fundamentação da linguagem. Ademais, assim como apresentado em (GUIZZARDI; GRAÇAS; GUIZZARDI, 2011) pretende-se formalizar através de procedimentos formais a sequência de passos para construção dos padrões nos modelos OntoUML e propor uma implementação destes padrões em uma ferramenta para construção de modelos em OntoUML.

1.4 OBJETIVOS DA PESQUISA

Este trabalho tem como objetivo geral definir um Catálogo de Padrões de OntoUML, bem como os procedimentos para construção destes padrões e implementá-los em uma ferramenta de modelagem. Esse objetivo geral pode ser detalhado nos seguintes objetivos específicos:

- Construção do Catálogo de Padrões;
- Catalogação dos procedimentos para construção dos modelos OntoUML usando os padrões definidos nesse catálogo;
- Implementação dos padrões em uma ferramenta de modelagem.

1.5 MÉTODO DE PESQUISA

Este trabalho foi conduzido de acordo com os seguintes passos:

i. *Análise dos Padrões já identificados de OntoUML:* Analisar os padrões já definidos para linguagem, como os mostrados em (GRAÇAS, 2010), (GUIZZARDI; GRAÇAS; GUIZZARDI, 2011) e (GUIZZARDI, 2005);

ii. *Análise das regras de fundamentação:* Analisar as regras de fundamentação de UFO procurando identificar o comportamento e a interdependência dessas regras;

iii. *Formalização dos procedimentos:* Uma vez identificados os padrões da linguagem, formalizar os procedimentos para construção destes padrões;

iv. *Implementação dos padrões:* Implementar os padrões identificados em uma ferramenta de modelagem de forma automatizada e prática;

v. *Avaliação da abordagem:* Avaliar o uso dos padrões na construção de um modelo OntoUML real. Testar a implementação dos padrões na ferramenta implementando um processo de detecção automática de incompletudes em modelos OntoUML e a correção dessas incompletudes utilizando os padrões criados;

vi. *Escrita da Dissertação:* os resultados obtidos durante a execução dos passos anteriores foram documentados nesta dissertação.

1.6 ORGANIZAÇÃO DA DISSERTAÇÃO

Neste capítulo inicial foram apresentadas as principais ideias desta dissertação, descrevendo o contexto de aplicação, motivações, objetivos e metodologia de pesquisa. Além desta introdução, este trabalho é composto pelos seguintes capítulos:

Capítulo 2 (Ontologias e Padrões Ontológicos): apresenta uma visão geral sobre o referencial teórico deste trabalho. É apresentado os conceitos mais importantes para o completo entendimento do trabalho.

Capítulo 3 (Padrões OntoUML): esse capítulo apresenta o processo de extração dos padrões das regras de fundamentação de UFO, a catalogação destes padrões em termos de fragmentos de OntoUML.

Capítulo 4 (Procedimentos para a Construção de Modelos OntoUML): nesse capítulo é apresentado um Processo Geral para Construção de Modelos em OntoUML. Este procedimento é subdividido em procedimentos específicos, como Procedimentos Básicos (relativos a tipos de objetos) e Procedimentos Dependentes (relativos a relacionamento).

Capítulo 5 (Avaliação da Abordagem): esse capítulo apresenta uma avaliação da abordagem utilizada. Apresentamos a implementação do catálogo de padrões em uma ferramenta de modelagem, uma avaliação do uso dos padrões em um exemplo real de modelagem e por fim algoritmo para verificação e correção de incompletudes em modelos OntoUML utilizando os padrões da linguagem.

Capítulo 6 (Considerações Finais e Perspectivas Futuras): apresenta as considerações finais do trabalho, ressalta suas contribuições e discute propostas de trabalhos futuros para continuidade e aprimoramento do trabalho.

CAPÍTULO 2. ONTOLOGIAS E PADRÕES ONTOLÓGICOS

2.1 ONTOLOGIAS

O estudo filosófico do *ser* enquanto *ser* (*being qua being*) possuí diferentes sentidos nas diversas comunidades da ciência (GUARINO; OBERLE; STAAB, 2009). Baseado na clássica definição de Gruber sobre o termo, temos que uma ontologia é na computação considerada como "*an explicit specification of a conceptualization*" - uma especificação explicita de uma conceituação (GRUBER, 1993). Em outras palavras, podemos interpretar o termo Ontologia como uma conceituação formal e consensual de uma conceituação (e.g. domínio de discurso). Essa interpretação expressa a defesa da ideia de que uma ontologia é uma visão compartilhada de várias partes sobre um determinado domínio, ou seja, um consenso ao invés de uma visão individual.

Dentre os sentidos empregados para os estudos sobre Ontologia, o mais pertinente para este trabalho se faz entre a abordagem filosófica frente ao sentido computacional empregado para o termo. Enquanto para Filosofia, Ontologia se apresenta como o estudo do seres quanto a sua existência e a relação entre as coisas que os permeiam (a relação entre os indivíduos e as coisas que existem), o sentido computacional empregado para o termo parte de uma visão mais recente na qual estas discussões são aplicadas na forma de um tipo especial de informação, ou seja, um artefato computacional (modelo, diagrama) enriquecido semanticamente e baseado em uma sequência de regras lógicas que definem o comportamento do modelo e as regras de instanciações dos tipos (GUARINO; OBERLE; STAAB, 2009).

Como discutido por Guarino em seu famoso trabalho "What is an Ontology?" (GUARINO; OBERLE; STAAB, 2009), a Ciência da Computação refere-se a ontologia como um objeto computacional (ontologia, com "o" minúsculo, no sentido de um artefato). Nesta visão, o conceito de existência possuí um sentido mais pragmático: "*for Artificial Intelligent systems, what 'exists' is that which can be represented*" – para sistemas artificiais inteligentes, o que 'existe' é aquilo que pode ser representado.

Embora muitos trabalhos se autodeclarem sistemas baseados em ontologia (*ontology-based applications*), a maioria não aplicam os métodos formais defendidos por Guarino, i.e. aplicações com modelos cujo as relações sejam estruturadas satisfazendo postulados bem definidos (uma fundamentação teórica) (GUARINO, 2009), nem tão pouco possuem uma visão compartilhada acerca do domínio modelado. Nestes casos, a essência da definição de ontologia (um consenso ao invés de um visão individual) não se concretiza. Neste trabalho, nós defendemos a modelagem conceitual bem-fundamentada baseada em Ontologias de Fundamentação (i.e. modelagem ontológica), a qual contribuí para construção de modelos semanticamente mais ricos, bem como, com maiores possibilidades de compartilhamentos (GUIZZARDI, 2005). Desta forma, destacamos a diferença entre modelagem conceitual, modelagem conceitual bem-fundamentada e modelagem ontológica quanto ao emprego de questões ontológicas no processo de modelagem. Enquanto a modelagem conceitual lida com a identificação dos conceitos de um domínio e o relacionamento entre eles, a modelagem conceitual bem-fundamentada é guiada por questões de natureza ontológica (i.e. modelagem baseada em ontologias), já a modelagem ontológica é baseada no uso ontologias de fundamentação.

Neste trabalho pretendemos trabalhar com uma abordagem baseada em padrões para construção de modelos conceituais bem-fundamentados, logo escolhemos a linguagem OntoUML para servir de base para nossos estudos. Está linguagem, como discutido anteriormente, é baseada na ontologia de fundamentação UFO. Em ambos os casos, temos um corpo de trabalho que tem sido usado em larga escala (GUIZZARDI et al., 2015) e que demonstram os benefícios tanto de OntoUML quanto de UFO para comunidade de Modelagem Conceitual e Ontologias. Ademais, temos que o professor orientador deste trabalho, Dr. Giancarlo Guizzardi, seja o criador de UFO e OntoUML, o que nos traz mais familiaridade com as teorias usadas para formalizar estas teorias. Em sequência apresentamos com mais detalhes UFO.

2.2 A ONTOLOGIA DE FUNDAMENTAÇÃO UNIFICADA

Em (GUIZZARDI, 2005) é apresentado a Unified Foundational Ontology (UFO) e proposto OntoUML como um linguagem para construção de modelos conceituais bem fundamentos baseada nas distinções ontológicas e teorias filosóficas apresentadas em UFO. A Figura 1 na sequência é apresentado um fragmento das distinções ontológicas propostas por UFO. Para

uma descrição mais detalhada sobre UFO e OntoUML, podemos fazer referência a (GUIZZARDI, 2005).

É importante notar que a figura apresenta uma taxonomia de universais (*Universals* – conceitos ou relações mais básicos de um determinado domínio), o que significa que as instâncias dessas classes serão tipos nos modelos OntoUML. Por exemplo, uma instância da classe Kind poderia ser a classe Kind *Person* (Pessoa) em um modelo. Já as instâncias dessa classe Kind *Person* poderiam ser *John* e *Mary*[1].

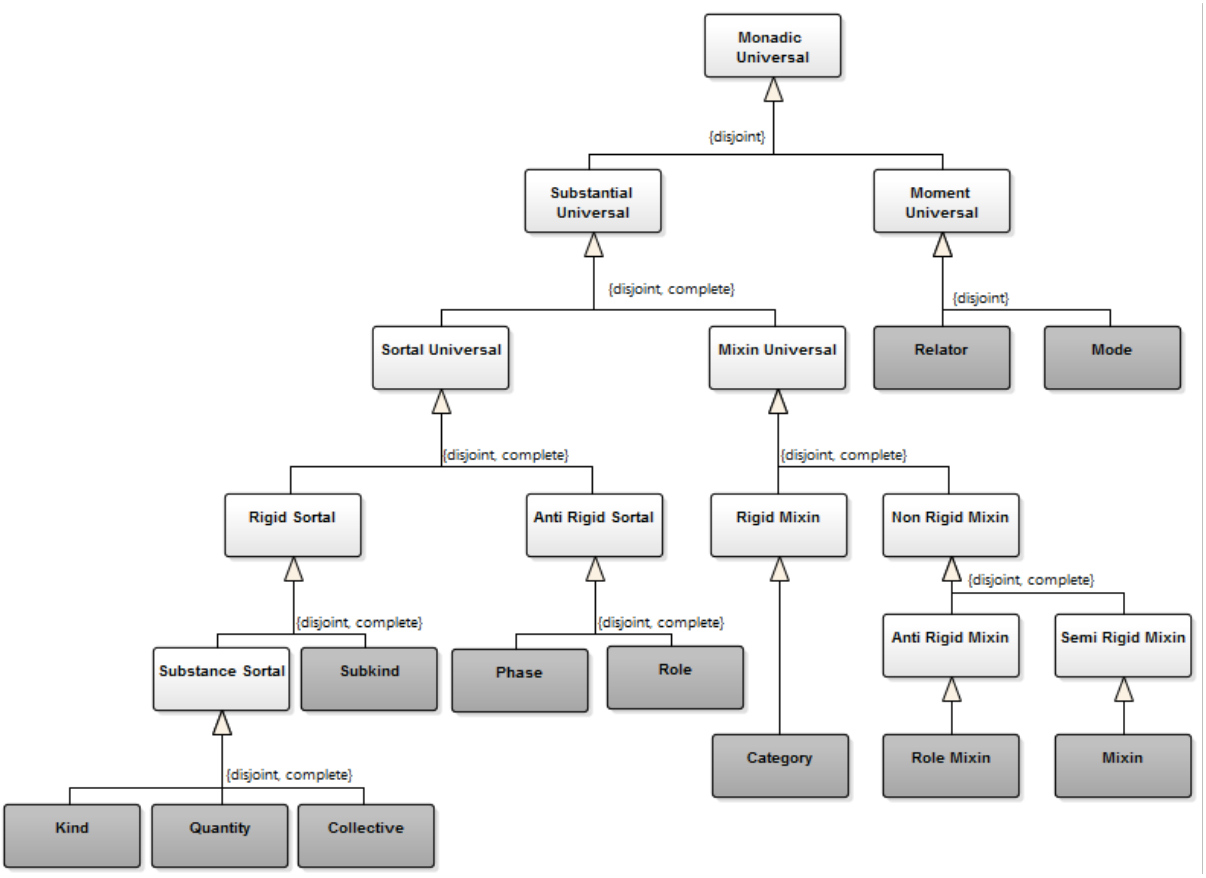

Figura 1 - Fragmento do Metamodelo de UFO – Adaptado de (GUIZZARDI, 2005)

No topo da figura temos o tipo *Monadic Universal*[2]. Na Lógica e Matemática, o termo *Monadic* determina o que predica sobre um ente. Já o termo *Universal*, tem-se seu sentido

[1] Alguns dos exemplos serão usados em inglês por serem extraídos de outros trabalhos.
[2] Os termos técnicos usados neste trabalho serão reproduzidos em inglês porque não existem traduções para todos esses termos consolidados na literatura em português e porque os termos são frequentemente usados em inglês na literatura cientifica. Para uma tradução parcial desses termos, o leitor interessado pode se referir a http://www.inf.ufes.br/~gguizzardi/IEEE2008.pdf.

ancorado na metafísica e significa as características e qualidades que são recorrentes em uma multiplicidade de indivíduos. Logo, *Monadic Universal* pode ser entendido como os predicados únicos das características e qualidades que são recorrentes em diversos indivíduos, como exemplos podemos citar: Pessoa, Pensamento, Cachorro, Sorriso etc.

A seguir temos a primeira distinção na taxonomia. A distinção entre *Substantial Universal* e *Moment Universal* é caracterizada quanto a dependência de suas instâncias. Em resumo, *Substantials (i.e., instâncias de Substantial Universals)* (também conhecido como *Endurant*, *Continuant* ou *Objeto*) são objetos concretos do cotidiano tais como: Gatos, Pessoas, Casas, Carros etc. Por outro lado, Moments (i.e., *instâncias de Moment Universals)* são entidades essencialmente inerentes a outro indivíduo, ou seja, toda instância de *Moment* está relacionado (é inerente - intrínseco) a alguma outra instância. Como exemplos temos: Peso, Pensamento, Carga Elétrica, Sorriso ou Dor de Cabeça. Como pode ser visto, enquanto *Substantial* existem no mundo de forma independente, *Moments* são coisas cujo a existência é dependente do suporte de outras entidades.

Através do ramo dos *Substantial Universals* nós temos a distinção entre *Sortais* e *Mixins*. Dos *Sortais* temos que é a classificação das coisas que obedecem a um princípio uniforme de identidade (ou que provê algum), enquanto que os *Mixins* são classes com a finalidade de agrupar tipos que implicam diferentes princípios de identidade para suas instâncias. Para ambos os casos temos subtipos sendo distinguidos quanto a sua meta-propriedade de rigidez.

A meta-propriedade de rigidez dita o comportamento das instâncias de um determinado tipo em decorrência das mudanças que suas instâncias possam sofrer no modelo (e.g. mudanças de atributos de algum tipo, novas instâncias, destruição de instâncias). Em outras palavras, entende-se por estas mudança as possíveis variações modais entre instanciações do modelo. Em resumo, temos que um tipo *T* é rígido se e somente se para toda instância *x* de *T*, *x* seja necessariamente uma instância de *T* em todos os mundos possíveis. Em contraste a meta-propriedade de ser rígido, temos a meta-propriedade de ser anti-rígido. Temos que um tipo *T'* é anti-rígido se e somente se para toda instância *y* de *T'* exista um mundo em que *y* não seja instância de *T'*. Em outras palavras, é possível que *y* deixe de ser do tipo *T'* sem deixar de existir (i.e. sem perder sua identidade) (GUIZZARDI, 2005).

Um exemplo desse distinção pode ser visto ao se contrastar o tipo rígido *Person* com o tipo anti-rígido *Living Person* (Pessoa Viva) e *Deceased Person* (Pessoa Morta) como ilustrado na Figura 2 abaixo.

Figura 2 - Exemplo de instâncias rígidas e anti-rígidas[1]

Na figura acima temos um primeiro instante *w* onde ambas as instâncias (círculos verde e laranja) pertencem ao tipo rígido *Person* e ao tipo anti-rígido *Living Person*. Em um segundo instante *w'* temos que a instância laranja muda do tipo *Living Person* para o tipo *Deceased Person*, entretanto, sem deixar de ser do tipo *Person*.

Outro importante ponto a ser notado é quanto a acessibilidade dos mundos. UFO pressupões uma relação de acessibilidade universal entre os mundos, ou seja, não há restrições quanto as possibilidades de navegação entre os mundos, deixando a critério do modelador definir como as mudanças de mundo podem ocorrer. Em outras palavras, não é restrito pela linguagem a visão linear do tempo, sendo possível por exemplo que após *w'*, o mundo *w* fosse novamente o mundo corrente, resultando no fato da instância laranja navegar de *Deceased Person* para *Living Person*. Essa decisão por parte da fundamentação de UFO torna genérica a

[1] Figura extraída dos slides de aula do professor Giancarlo Guizzardi: http://guizzardi.panrepa.org - Acessado em 17/01/2016

interpretação de acessibilidade proporcionando maior flexibilidade da linguagem quanto a isso.

Embora UFO prescreva uma relação de acessibilidade universal entre mundos, a restrição desse comportamento pode ser feita utilizando regras de restrições. Dentre os trabalhos que objetivam isto, temos o apresentado em (GUERSON, 2015) o qual apresenta um conjunto de regras temporais para lidar com aspectos da fundamentação em modelos OntoUML.

De volta a Figura 1, temos que os tipos *Sortais* rígidos são especializados em *Substance Sortal* e *Subkind*. Os *Substance Sortals* são tipos rígidos e que que proveem princípio de identidade para suas instâncias (e.g. *Person*). Já os *Subkinds* são tipos rígidos que carregam a identidade provida por um *Substance Sortal* (e.g. *Man*, *Woman*). De acordo com as regras de fundamentação de UFO, temos que *i)* todo indivíduo deve obedecer um princípio de identidade e *ii)* um indivíduo não pode obedecer a mais de um princípio de identidade, logo, temos que cada instância em um modelo conceitual é necessariamente instância de exatamente um *Substance Sortal* (GUIZZARDI, 2005).

Adentrando a definição de *Substance Sortal*, temos que estes são os tipos mais importantes do modelo conceitual. Em UFO temos três tipos de *Substance Sortal*, sendo: *Kind*, *Quantity* e *Collective*. O primeiro representa tipos cujo as instâncias sejam complexos funcionais (*functional complexes* – entidades compostas por partes que executam diferentes papéis no contexto de seu todo), como: *Pessoa, Cachorro, Cadeira* e *Carro*. Já o segundo representa tipicamente tipos que na linguagem natural nos referimos a quantidades, porções, tais como: *Ouro, Quantidade (maximal) de água em um copo* e *Areia*. Por último, temos que os Collectives representam instâncias de coletivos de complexos com uma estrutura uniforme, tais como: *Baralho, Floresta, Grupo de Pessoas* e *Pilha de Tijolos*.

Dentre os *Sortais* anti-rígidos, temos dois tipos, *Phase* e *Role*. Em ambos os casos temos uma classificação dinâmica das instâncias desses tipos, i.e. as instâncias podem mover para dentro e para fora desses tipos sem perder sua identidade (como no exemplo da Figura 2). Entretanto, enquanto a origem da mudança de um tipo *Phase* ocorre por alguma mudança intrínseca às instâncias daquele tipo, as mudanças de uma instância de *Role* ocorrem pela mudança de suas propriedades relacionais. Para exemplificar essa distinção, podemos contrastar as *Phases Child, Adolescent* e *Adult* de um tipo *Person* com os Roles *Student* e

Husband e *Wife* também do tipo *Person*. Como podemos ver, em ambos os casos os *Phases* e *Roles* do tipo *Person* são passíveis de possuírem instâncias ou não (i.e. são anti-rígidos). Embora as instâncias dos *Phases* de *Person* possam mudar em decorrência de uma mudança intrínseca do tipo (e.g. idade), a instanciação ou não dos *Roles* do tipo *Person* precisam de uma relação com instâncias de outro tipo para que isso possa acontecer. A título de exemplo, poderíamos dizer que para assumir o *Role* de *Student* uma instância *p* de *Person* deveria relacionar-se com um outro tipo *Education Institution*, além disso, a exclusão dessa relação seria suficiente para que *p* deixa-se de instanciar o *Role Student*.

Análogo ao *Role Student*, os *Roles Husband* e *Wife* também são relacionalmente dependentes. Em resumo, diferentemente do tipo *Phase*, *Roles* possuem a propriedade de serem relacionalmente dependentes. Formalmente temos que um dado tipo *T* é relacionalmente dependente de um outro tipo *P* via uma relação *R*, se e somente se para cada instância *x* de *T* existir uma instância *y* de *P* tal qual *x* e *y* estejam relacionados por *R*. Em outras palavras, temos que todo *Role* no modelo conceitual precisa estar relacionando por uma associação com outro tipo direta ou indiretamente (e.g. através dos seus subtipos).

Como defendido em (GUIZZARDI, 2005), *Phases* constituem uma partição do *Substance Sortal* o qual eles especializam. A está particularidade, que não se manifesta apenas em *Phases* como veremos nas próximas seções, chamamos de Partição (*Partition*). Na Modelagem Conceitual uma partição é caracterizada pela divisão de um tipo *K* em subtipos de *K*, sendo esses disjuntos entre si e não havendo instanciação isolada para *K*. Desse fato, resulta-se que em cada mundo possível, toda instância de *K* é necessariamente instância de exatamente um dos Phases *Pi* de uma Phase Partition. Consequentemente, temos que necessariamente Partitions são disjuntas (não existe interseção entre *Pi's*) e completas (o tipo *K* não pode ser instanciado diretamente, i.e. *K* se torna abstrato para essa partição).

Como supracitado, uma das distinções da Figura 1 é entre *Substantial Universals* e *Mixin Universals*. Enquanto *Substantial Universals* são tipos que proveem ou carregam identidade para suas instâncias, *Mixins* são tipos com a finalidade de agrupar outros tipos com diferentes princípios de identidades (ou outros *Mixins*, os quais por sua vez, já agrupam tipos de diferente princípios de identidade). Assim como para os *Sortais*, os *Mixins* também são diferenciados quanto sua meta-propriedade de rigidez. Entretanto, para *Mixins*, temos, além de rígidez e anti-rígidez, a meta-propriedade de semi-rígidez.

Em resumo os *Mixin Universals* agrupam tipos que compartilham alguma propriedade comum entre si, mas que não possuem um princípio de identidade uniforme entre todas as suas instâncias. A título de exemplo podemos constatar que enquanto um tipo *Person* possuí instâncias de seres humanos, um tipo *Artificial Agent* poderia possuir instâncias como supercomputadores com avançada inteligência artificial, o que poderia classifica-los como seres racionais. Nesse sentido, podemos criar um tipo de *Mixin* chamado *Rational Entity* com a finalidade de agrupar em um só tipo instâncias de tipos racionais, i.e. tanto o tipo *Person* quanto o tipo *Artificial Agent*. A Figura 3 abaixo ilustra esse exemplo.

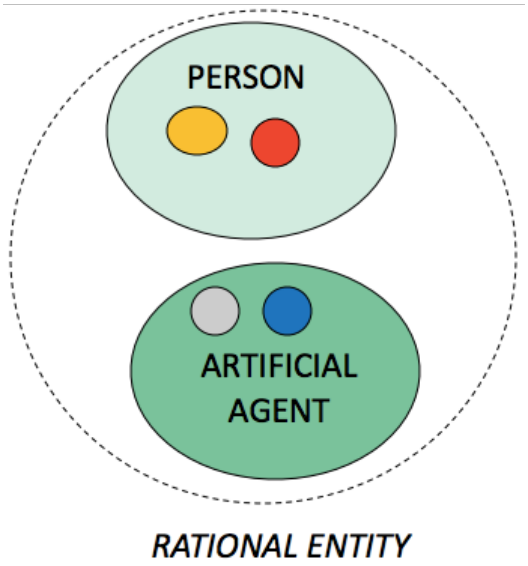

Figura 3 - Exemplo de Mixin agrupando tipos de identidade diferentes[1]

Como pode ser visto na figura acima, ao se discursar sobre o tipo *Rational Entity*, estamos nos referindo ao conjunto de instâncias tanto de *Person* quanto de *Artificial Agent*.

O *Mixin* que agrupa instâncias que sempre pertencerão a sua extensão (i.e. o conjunto de todas as suas instâncias em um dado mundo *w*) é chamado *Category*. Sendo um tipo rígido, *Categories* necessariamente possuem instâncias que nunca deixaram de pertencer a sua extensão.

Por outro lado, o *Mixin Universal* anti-rígido chamado *RoleMixin* se comporta de forma diferente. Sendo um tipo anti-rígido, para toda instância da extensão de um *RoleMixin* ela

[1] Figura extraída dos slides de aula do professor Giancarlo Guizzardi: http://guizzardi.panrepa.org - Acessado em 17/01/2016

deixa de instanciar esse *RoleMixin* em um mundo *w* possível. Além dessa meta-propriedade, assim como no caso de *Roles*, os *RoleMixins* também possuem a particularidade de serem relacionalmente dependentes. Isso implica que além de *RoleMixins* agruparem tipos anti-rígidos (e.g. *Role* e *RoleMixin*), eles também precisam estar relacionados a outros tipos através de associações que representam as condições relacionais de instanciação de suas instâncias (dependência relacional), tais como relações materiais e de mediação (a definição desses termos será feita mais adiante no texto).

Por fim, o *Mixin Universal* classificado como semi-rígido também é chamado *Mixin*. Por definição, um *Universal* é um semi-rígido se e somente se for um não-rígido mas não um anti-rígido. Para este tipo temos que necessariamente ele precisa possuir tipos que sejam rígidos e outros que sejam anti-rígidos.

Embora todas essas distinções sirvam para os *Substantial Universal*, também temos o tipo *Moment Universal*. Para esse, temos uma distinção entre *Relator Universals* e *Mode Universals*. *Relators* são existencialmente dependentes (caracteriza a relação na qual o indivíduo dependente existe somente se o outro indivíduo do qual aquele depende também existir) de ao menos duas outras entidades (necessariamente distintas). Exemplos incluem *Casamento*, *Contrato* e *Matrícula*. *Modes*, em contraste, se apresentam como um tipo intrínseco e sendo existencialmente dependente de exatamente uma entidade, e.g. *Habilidade*, *Dor-de-cabeça*, *Sintoma*.

Devido a dependência existencial dos *Relators*, temos que *Relators* precisam estar conectados via uma *Mediation Relation* (tipo de relação discutida na sequência) com alguma outra entidade. Para exemplificar esse comportamento, vamos supor um casamento entre dois indivíduos chamados *John* e *Mary*. A Figura 4 abaixo idealiza um modelo com estes indivíduos.

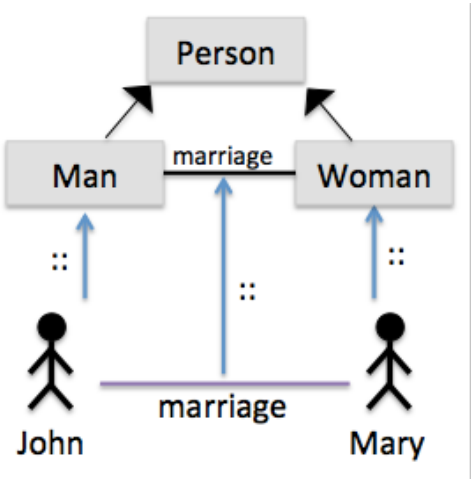

Figura 4 - Exemplo de um modelo de casamento

Na figura a cima (a qual serve apenas de ilustração para o exemplo que se segue) podemos ver um modelo de um típico casamento (*marriage*) entre um homem (*Man*) e uma mulher (*Woman*). Note que o sinal de instanciação (*::*) é usado para denotar a relação de *instance of*, i.e. a relação entre uma instância *x* e o tipo *T* cuja essa instância está instanciando (*x instance of T*).

Neste exemplo temos apenas a informação de que *John* está casado com *Mary*, o que para muitos Sistemas de Informação é suficientes para analises simples, entretanto, a modelagem conceitual bem fundamentada nos permite ir além e verificar a essência por detrás destas informações. Neste sentido UFO prescreve a criação de *Relators* para as relações materiais entre tipos (e.g. a relação *marriage*). Dessa forma, o uso de um *Relator Marriage* induz a criação dos papéis de cada tipo (*Roles*) em cada lado da relação. Essa particularidade de UFO (a qual é herdada por OntoUML) é tida como uma boa prática de modelagem e faz parte das vantagens do uso de uma Ontologia de Fundamentação para criação de modelos conceituais bem fundamentados (GUIZZARDI, 2005). Além disso, este comportamento reforça o argumento de que OntoUML (linguagem baseada em UFO) seja uma linguagem baseada em padrões, como veremos mais a frente. Para ilustrarmos a visão de UFO deste exemplo, introduzimos a Figura 5 abaixo que retrata como ficaria este modelo descrito na linguagem OntoUML.

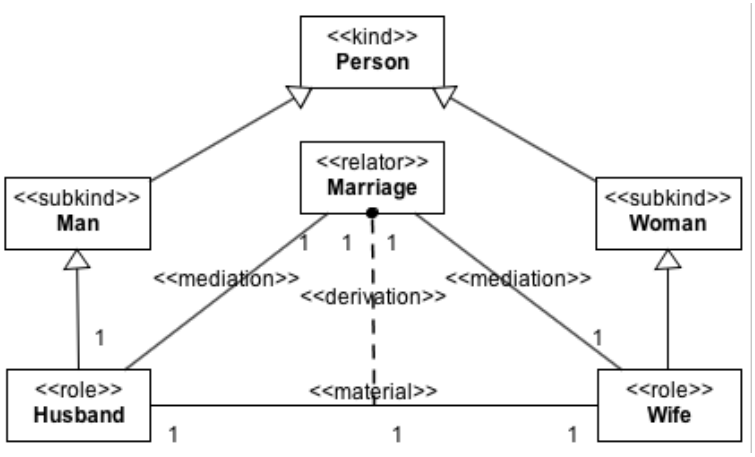

Figura 5 - Modelo de casamento em OntoUML

Como pode ser visto, a presença do Relator *Marriage* induz a necessidade de especializar os tipos *Man* e *Woman* em *Husband* e *Wife* respectivamente. Além disso, podemos ver que agora temos uma base de fundamentação por detrás de nosso modelo. Ou seja, estamos explicitando os papéis de cada entidade nos relacionamentos nos quais elas participam, gerando assim mais informação sobre domínio modelado.

Por fim temos as dentições ontológicas importas por UFO as relações. Na Figura 6 abaixo temos a classificação hierárquica destas relações.

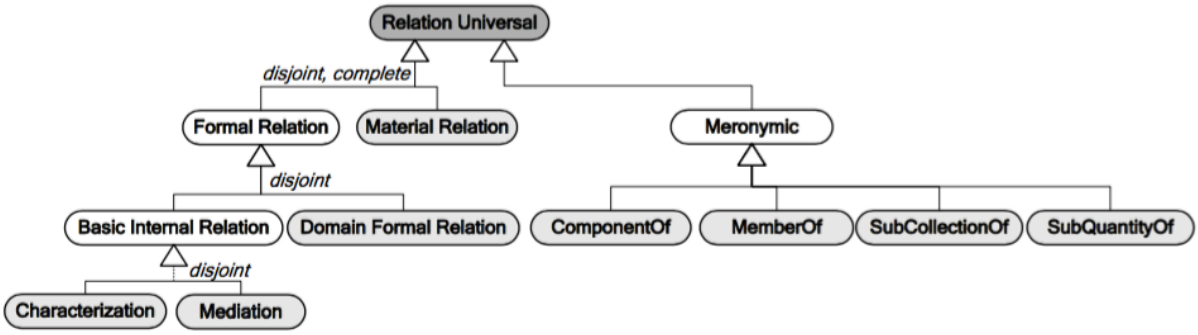

Figura 6 - Fragmento de UFO - Extraído de (ZAMBORLINI, 2011)

O tipo *Material Relation* se aplica às relações que dependem de alguma entidade para existir. Por exemplo, a relação apresentada no exemplo da Figura 5 (a relação *marriage*) que só existe enquanto existir o *Relator Marriage* entre *John* e *Mary*. Contrariamente, as relações do tipo *Formal Relation* valem pela simples existência dos indivíduos relacionados. Por exemplo, a relação parte de entre João e seu Cérebro vale sempre que ambos existirem.

Já as relações do tipo *Formal* podem ser ainda classificadas como Basic Internal Relation ou Domain Formal Relation. O tipo Basic Internal Relation aplica-se a relações formais internas

ditas de dependência existencial (caracteriza a relação na qual o indivíduo dependente existe somente se o outro indivíduo do qual aquele depende também existir) e se subdivide em *Characterization*, que se aplica à relação de inerência (*inheresIn*) que define os indivíduos do tipo *Moment*, e *Mediation*, que se aplica à relação de mediação (*mediates*) que define os indivíduos do tipo *Relator*. Além desses tipos, também temos o caso específico da relação *Derivation* que ocorre entre uma relação *Material* e o tipo *Relator* que originou esta relação. Diferentemente das demais relações, a relação *Derivation* é diagramada como uma linha pontilhada.

As relações do tipo Domain Formal Relation se aplicam às relações formais que são especificas de domínio e que, por isso, não são representadas entre os tipos de indivíduos da UFO, mas entre os conceitos específicos de domínio, assim como as do tipo *Material Relation*.

Por fim, o tipo *Meronymic* refere-se as relações de todo-parte (*partOf*). Esse tipo é subdivido nas relações *ComponentOf*, *MemberOf*, *SubCollectionOf* e *SubQuantityOf*, as quais serão discutidas em sequência.

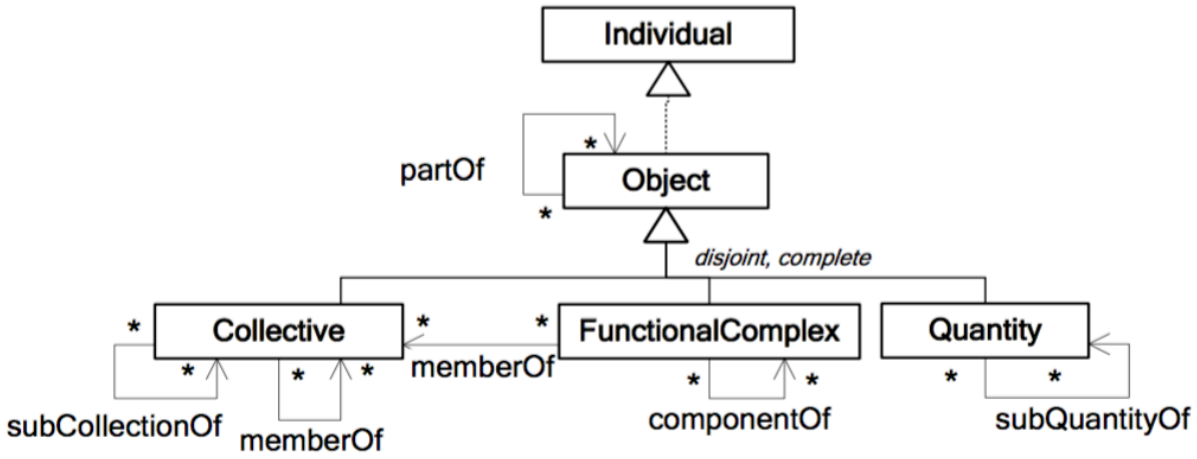

Figura 7 – Fragmento de UFO sobre as relações Meronymics (ZAMBORLINI, 2011)

Com pode ser visto na figura a cima, as relações *Meronymics* possuem restrições diferente das demais, como se segue: *componentOf* vale entre complexos funcionais, *memberOf* vale entre complexos funcionais e coleções, ou coleções entre si, *SubCollectionOf* vale entre coleções e *SubQuantityOf* vale entre quantidades. Nas seções seguintes explicaremos com mais detalhes como cada uma dessas relações podem ser usadas e suas demais restrições.

Aqui finalizamos nossa discussão sobre os conceitos mais importantes de UFO os quais serão usados ao longo deste trabalho. É importante notar que nós apenas discutimos sobre uma fração dos tipos presentes em UFO, i.e. UFO-A, a qual lida com os aspectos mais gerais da fundamentação ontológica. Existem muitos outros não tratados aqui, como os tipos discutidos em UFO-B (aspectos ontológicos sobre eventos) (GUIZZARDI et al., 2013) e UFO-C (aspectos ontológicos sociais) (BRINGUENTE et al., 2008) (GUIZZARDI et al., 2011), entretanto estes estão fora do escopo deste trabalho. Escolhemos apenas trabalhar com UFO-A porque, como veremos a seguir, a linguagem OntoUML (a qual iremos adotar neste trabalho), até o momento, utiliza apenas os construtos definidos em UFO-A.

2.3 ONTOUML

Como já mencionado, a linguagem OntoUML foi primeiramente apresentada em (GUIZZARDI, 2005) como uma extensão da versão 2.0 da Unified Modeling Language (UML). Com essa extensão, OntoUML acrescente várias distinções e axiomas ao metamodelo de UML. Por exemplo, OntoUML estende o construto *Class* de UML para capturar as distinções ontológicas de UFO (discutidas na seção anterior). Um fragmento do resultado desta extensão é mostrado na Figura 8 abaixo.

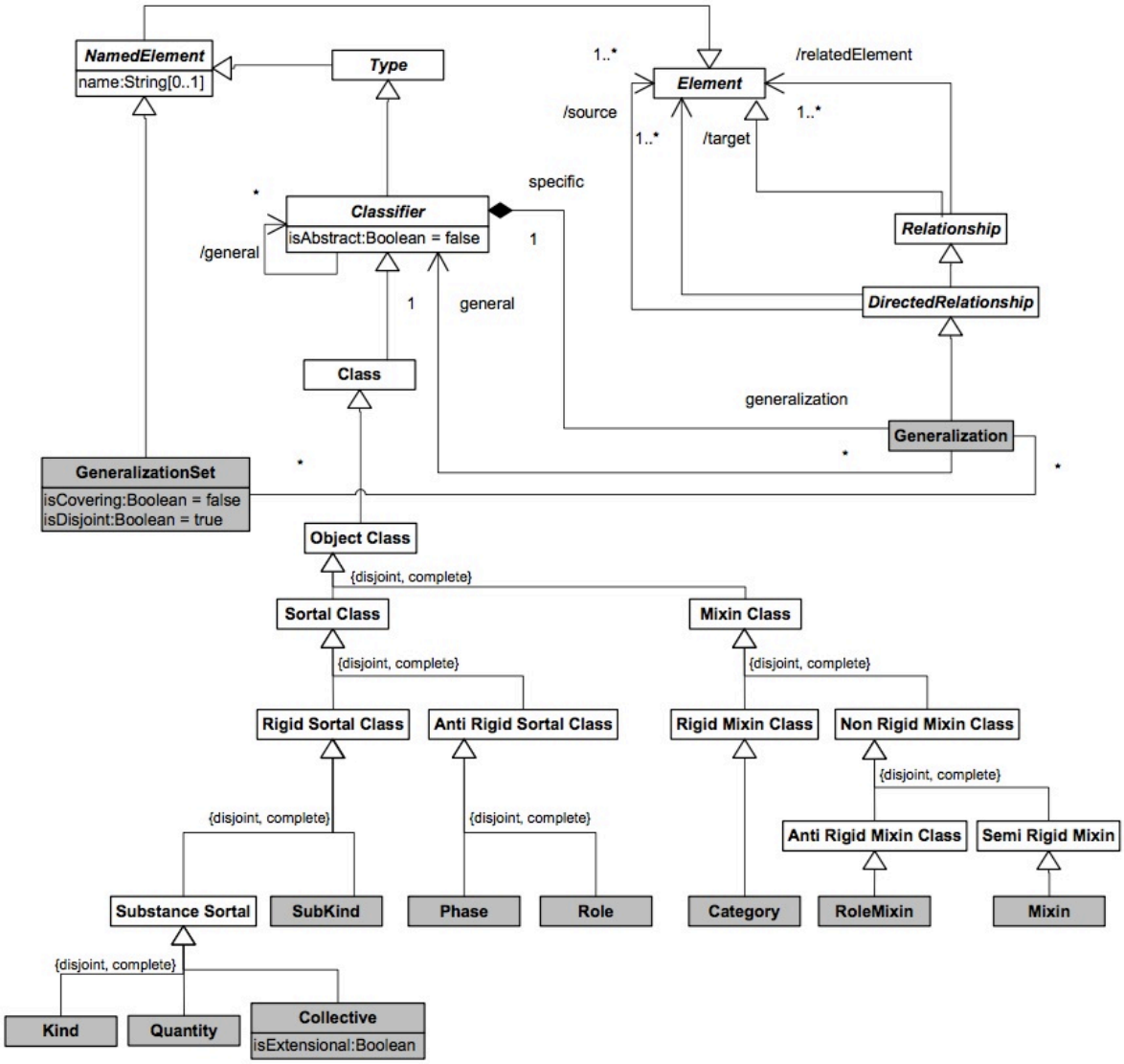

Figura 8 – Fragmento do metamodelo de OntoUML (GUIZZARDI, 2005)

Olhando para figura acima podemos ver que a maior mudança no metamodelo original de UML é a adição das distinções para o construto *Class*. Vale ressaltar que apenas as folhas deste tipo podem ser instanciados (destacado em escuro), uma vez que as outras classes são abstratas.

Uma das premissas de OntoUML é de manter consistência com as regras de UML. Em outras palavras, cada modelo OntoUML sintaticamente válido também será um modelo UML válido. Para que isso aconteça, os possíveis tipos instanciáveis de OntoUML são representados em UML como estereótipos de uma classe (veja Figura 4 como exemplo de um modelo OntoUML válido). Consequentemente, uma das restrições que OntoUML impõe para UML é a redução para apenas uma das possibilidades de estereótipo para uma dada classe.

Por fim, temos que inúmeros benefícios surgem do fato de OntoUML manter a estrutura sintática de UML inalterada. Podemos destacar o fato de um modelo OntoUML poder ser criado usando qualquer ferramenta de diagramação que suporte a sintaxe de UML 2.0. Além disso, ferramentas de analise de modelos também podem ser usadas, bem como linguagens para descrição de regras como a Object Constraint Language (OCL) que possibilita aplicar restrições que não seriam possíveis representar apenas com um diagrama (e.g. a restrições de atributos). Entretanto, ainda que possa-se modelar OntoUML utilizando ferramentas para UML, diversas técnicas mais modernas surgiram com o intuito de auxiliar os modeladores a construção de modelos conceituais bem fundamentados, as quais tornaram necessário a criação de um editor específico para OntoUML que pudesse comportar estas novas técnicas. Diante disto o OntoUML Lightweight Editor (OLED) surge como uma tentativa de unir todos estes esforços em um único ambiente.

2.4 OLED: UM EDITOR ONTOUML

O OntoUML Lightweight Editor[1] (OLED) (GUERSON et al., 2015) é um framework para o desenvolvimento de modelos OntoUML que provê inúmeras funcionalidades para auxiliar os modeladores na construção de modelos completos (representando todos os conceitos pertinentes ao domínio modelado) e corretos (alinhados e respeitando as regras de fundamentação de UFO).

Para prover este auxilio, OLED fornece as funcionalidades de formalização, validação, verificação, geração de código, verbalização de modelo e simulação, além da funcionalidade de diagramação. Nesta seção pretendemos mostrar uma visão geral de algumas das principais funcionalidades da ferramenta, além de destacar as funcionalidades existentes antes das alterações propostas neste trabalho. Ademais salientamos que as mudanças introduzidas por este trabalho na ferramenta é discutida em partes durante o texto, dessa forma buscamos apresentar ao leitor estas mudanças de forma gradativa e seguindo o fluxo do texto.

[1]OLED – Disponível em: https://github.com/nemo-ufes/ontouml-lightweight-editor/

Resumindo as funcionalidades da ferramenta temos a descrição apresentada na Figura 9 abaixo e extraída de (GUERSON et al., 2015), e que será discutida em sequência.

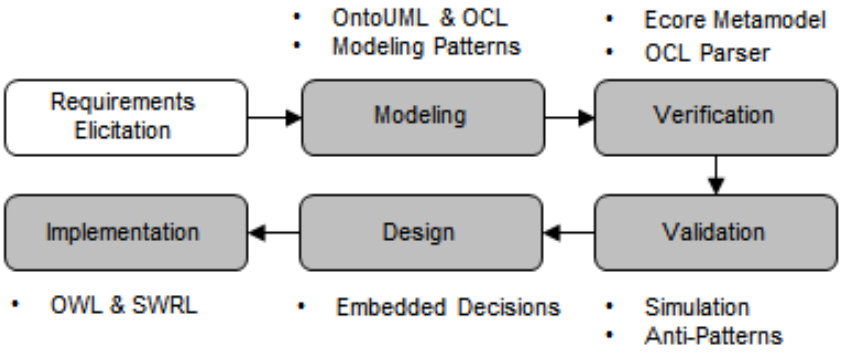

Figura 9 - Visão geral das funcionalidades do OLED

Sob o aspecto de modelagem, OLED prove várias funcionalidades básicas (e.g. centralização das classes, sobreposição de elementos, mudança de cor, exportar como imagem, zoom e manipulação do diagrama), como ilustrado na parte superior da Figura 10.

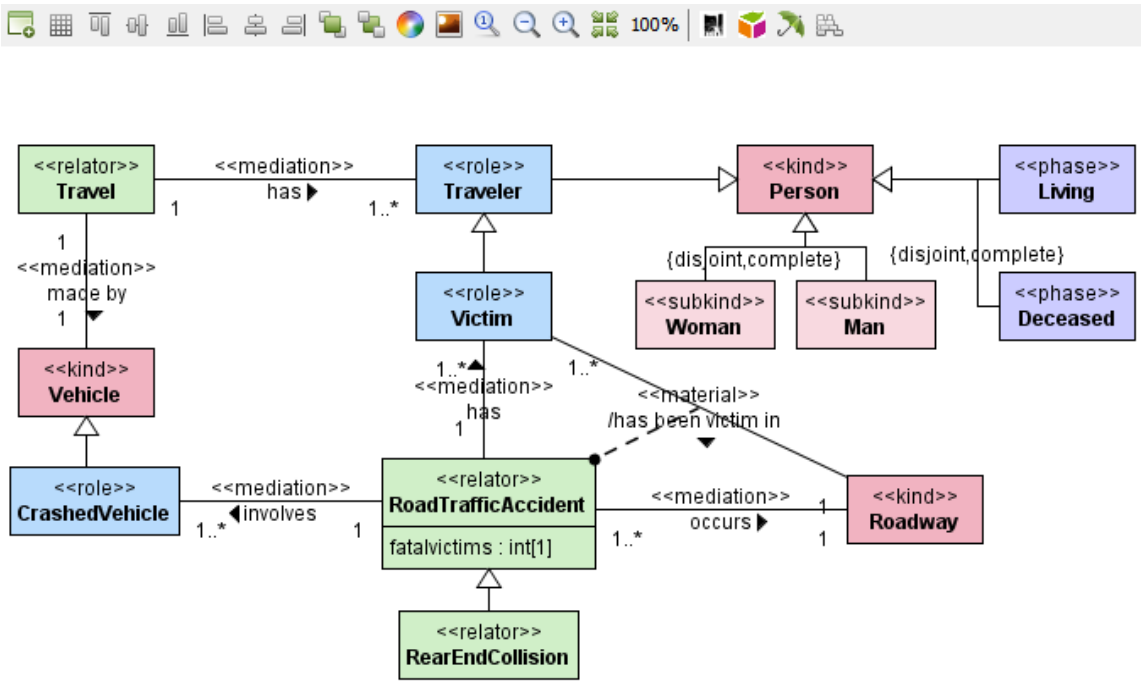

Figura 10 - Exemplo de diagrama feito no OLED (GUERSON et al., 2015)

Além disso, como já citado anteriormente, uma vez que nem todas as informações sobre um domínio podem ser definidas através de elementos diagramáticos, OLED também fornece a possibilidade de se criar regras OCL para restringir aspectos do modelo. Estas regras são embutidas no modelo OntoUML de forma que são também usadas em todas as outras funcionalidades da ferramenta. Desta forma, o modelador pode criar regras de restrição

clássicas (utilizando apenas OCL) além de restrições temporais (utilizando Temporal OCL - TOCL (GUERSON, 2015)), as quais ampliam a possibilidades de restrição para o modelo OntoUML (e.g. restrição da ordem de mudança dos Phases). Utilizando-se de um editor específico para isso, a ferramenta possibilita checagem de sintaxe da regra OCL, bem como a funcionalidade de auto-completar a escrita das regras envolvendo conceitos presentes no modelo.

Como funcionalidade de verificação, temos a possibilidade de analisar se a sintaxe do modelo está de acordo com as regras de fundamentação subjacente a OntoUML (i.e., as regras que constituem a axiomatização de UFO). Está análise é feita baseada em todas as restrições impostas por UFO em (GUIZZARDI, 2005). Entretanto, ainda que o modelo esteja sintaticamente válido, não podemos ter certeza se ele está representando de forma correta o domínio modelado. Neste caso, temos que o ato de validar um modelo OntoUML significa analisar se um dado modelo é o modelo pretendido para o domínio em questão. Neste contexto, algumas técnicas como simulação e checagem de anti-padrões se mostram extremamente úteis (GUERSON et al., 2015).

A validação via simulação (BRAGA; ALMEIDA; GUIZZARDI, 2010) possibilita ao usuário utilizar uma ferramenta gráfica para navegar entre instâncias aleatórias do modelo, verificando se o comportamento mostrado pela simulação é efetivamente um comportamento pretendido no domínio modelado. Essa funcionalidade também permite realizar testes buscando encontrar conjunto de instâncias que formam uma combinação não desejável para o domínios modelados e, em alguns casos, gerar regras automáticas para evitar que essas instâncias indesejadas possam se manifestar.

OLED também possuí uma ferramenta para checagem de anti-padrões. Neste contexto anti-padrões representam um conjunto recorrente de padrões propícios a gerar erros de modelagem (SALES; GUIZZARDI, 2015). Em outras palavras, anti-padrões são construções corretas de modelos (i.e. são sintaticamente válidos) que no entanto possibilitam instanciações não desejáveis do modelo. Os padrões identificados pela ferramenta são salientados ao usuário acompanhados de algumas propostas de retificação (SALES; GUIZZARDI, 2015).

Embora modelos conceituais sirvam para seres humanos entenderem e trocarem conhecimento entre si, ontologias operacionais tem a finalidade de serem entendidas por maquinas. Por isso, OLED provê diversas formas de se transformar de forma automática um modelo OntoUML para um linguagem de especificação lógica operacional, como a Web Ontology Language (OWL). Para está funcionalidade, a ferramenta possuí as seguintes transformações: Simple Transformation[1], Temporal Transformations (ZAMBORLINI, 2011) (GUIZZARDI; ZAMBORLINI, 2014) e OntoUML2OWL+SWRL (BARCELOS et al., 2013).

Como podemos perceber, OLED possuí diversas funcionalidades que apoiam o processo de construção de modelos conceituais em OntoUML, bem como transformações para ontologias operacionais, entretanto, são poucos os trabalho que utilizam abordagens baseada em padrões com o intuito de auxiliar os modeladores da linguagem. Nesse caso podemos citar o trabalho de (GRAÇAS, 2010), onde uma ferramenta para construção de modelos utilizando recorrente aplicações de padrões é proposta. Embora esse trabalho seja uma amostra do potencial das abordagens baseadas em padrão para modelagem conceitual, a ferramenta criada por Graças não está associada a uma ferramenta de edição como OLED, tornando complexa seu uso bem como reduzindo sua adoção pela comunidade.

Como mencionado anteriormente, este trabalho visa criar um catálogo de padrões para OntoUML bem como procedimentos para criação e uso destes padrões. Em decorrência desses objetivos e do ambiente proporcionado no OLED, pretendemos usá-lo como ferramenta base para nossas implementações. Na seção 5.1 discutimos a implementação realizadas na ferramenta para atingirmos estes objetivos.

2.5 CONSIDERAÇÕES FINAIS DO CAPÍTULO

O uso de Ontologias e suas técnicas na modelagem conceitual tem se mostrado um grande avanço para construção de modelos bem-fundamentados e formalmente estruturados. Entretanto, como discutimos aqui, o uso de Ontologias de Fundamentação como UFO tende a tornar o processo de modelagem inicialmente mais complexo para modeladores novatos.

[1] Transformação realizada pelo pesquisador Antognoni Albuquerque porém sem publicações sobre ela.

Nesse sentido, ferramentas como OLED que implementam diversas funcionalidades no auxílio a modelagem e validação de ontologias surge como um aliado na tentativa de simplificar o processo de modelagem de ontologias.

Dentro deste contexto, este trabalho visa utilizar os aspectos formais de UFO e sua linguagem de modelagem OntoUML para definição de uma serie de padrões de fundamentação ontológica (Foundational Ontology Patterns - FOPs) para servir como um catálogo de padrões em auxílio aos modeladores da linguagem. Entretanto, devido a complexidade inicial de aprendizado da linguagem e da difícil tarefa de extrair o conhecimento do domínio para construção de modelos conceituais de domínio, pretendemos discutir uma serie de procedimentos para confecção dos padrões apresentados no catálogo, com a finalidade de servirem de apoio para modeladores OntoUML. No próximo capítulo, apresentamos os padrões OntoUML já encontrados na literatura bem como os investigados neste trabalho.

CAPÍTULO 3. PADRÕES ONTOUML

3.1 EXTRAÇÃO DE PADRÕES ONTOLÓGICOS DE FUNDAMENTAÇÃO A PARTIR DE UFO-A

Na modelagem conceitual, padrões são blocos (fragmentos) de algum modelo que encapsula conhecimentos previamente discutidos. Estes fragmentos são considerados os mais efetivos para nomear, organizar e raciocinar sobre algum tipo de conhecimento (FALBO et al., 2013). Por definição, temos que um padrão descreve um problema particular recorrente que emerge de um contexto específico e que se apresenta como uma solução provada para este problema (BUSCHMANN; HENNEY; SCHMIDT, 2007). Neste sentido e como discutido em (RUY et al., 2011), os padrões ontológicos conceituais podem ser divididos em Foundational Ontology Patterns (FOPs) e Domain-Related Ontology Pattern (DROPs). Ambos são usados durante a fase de modelagem e focam somente no aspecto conceitual da modelagem, não se preocupando com tecnologias ou linguagens futuras de ontologias operacionais (implementação). FOPs são fragmentos reusáveis extraídos das regras das ontologias de fundamentação e uma vez que as ontologias de fundamentação definem conceitos genéricos e independentes de domínio, FOPs possuem a capacidade de poderem ser usados em qualquer domínios. Por outro lado, DROPs são fragmentos extraídos de ontologias centrais (RUY et al., 2011), capturando o conhecimento chave relacionado ao domínio destas ontologias e são aplicados por extensão (i.e., são inseridos no modelo e podem ser especializados ou generalizados) às ontologias de domínio.

Como discutido em (RUY et al., 2011), a derivação e aplicação de FOPs se dividem em dois tipos, os Structural FOPs e os Derivation FOPs. Enquanto o Structural FOP captura um template (fragmento) a partir da aplicação de regras da ontologia de fundamentação, os Derivation FOPs tratam sobre como novos conceitos podem emergir a partir de operações aplicadas ao modelo (GUIZZARDI, 2012). Entretanto, como este trabalho lida apenas com Structural FOPs iremos chama-los apenas de FOPs. Para mais informações sobre os Derivation FOPs deve-se analisar o trabalho de (REGINATO, 2015) onde é feita uma serie de análise sobre os tipos derivados em OntoUML bem como as respectivas regras de derivação.

Como já discutido, os FOPs emergem do resultado da aplicação de regras de fundamentação. Para exemplificar, vamos utilizar as regras de fundamentação extraídas de UFO abaixo e discutir como estas regras se relacionam derivando os padrões de OntoUML e suas variações.

Regras:

R1: Um *Relator* media ao menos dois indivíduos distintos;

R2: Uma relação de mediação (Mediation) é uma relação entre um *Relator* e um *Endurant*;

R3: Um *Role* é um tipo anti-rígido e precisa carregar um princípio de identidade fornecido por um *Substance Sortal*;

R4: Um *Role* é externamente dependente de um *Relator*.

Neste exemplo, temos que a combinação de R1 e R2 resultam no chamado Relator FOP, enquanto a combinação de R3 com R4 no Role FOP, ambos representados nas Figura 11 e Figura 12 respectivamente.

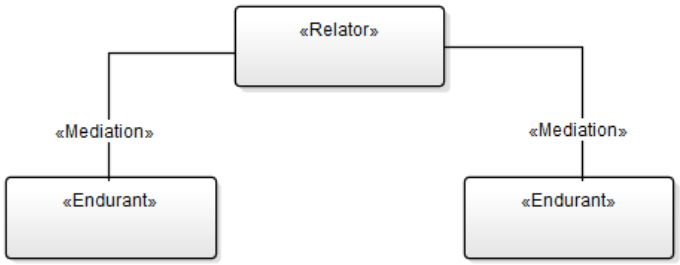

Figura 11 - Relator FOP

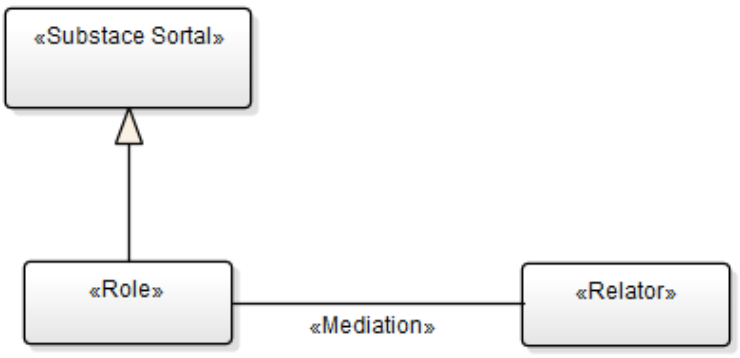

Figura 12 - Role FOP

É importante salientar que os tipos *Endurant* e *Substance Sortal* não são tipos diretamente instanciáveis em OntoUML. Eles aparecem aqui generalizando as possibilidades de instanciação para esta posição no modelo. Em outras palavras, estes tipos devem ser substituídos por um dos seus respectivos subtipos instanciáveis como ilustrado na Figura 1.

Como mencionado, tanto o Relator FOP quanto o Role FOP são padrões genéricos e possibilitam variações. Nesse sentido, poderíamos substituir o tipo estereotipado como *Endurant* do Relator FOP (Figura 11) usando a meta-classes específica *Role.* Isso eliminaria a necessidade de se aplicar R4, mas ainda assim teríamos mantida a obrigatoriedade de se aplicar a regra R3. Como consequência teríamos uma combinação dos dois padrões resultando em um novo padrão mais específico. Como discutido em (RUY et al., 2011), o resultado desta aplicação pode ser visto na Figura 13 abaixo.

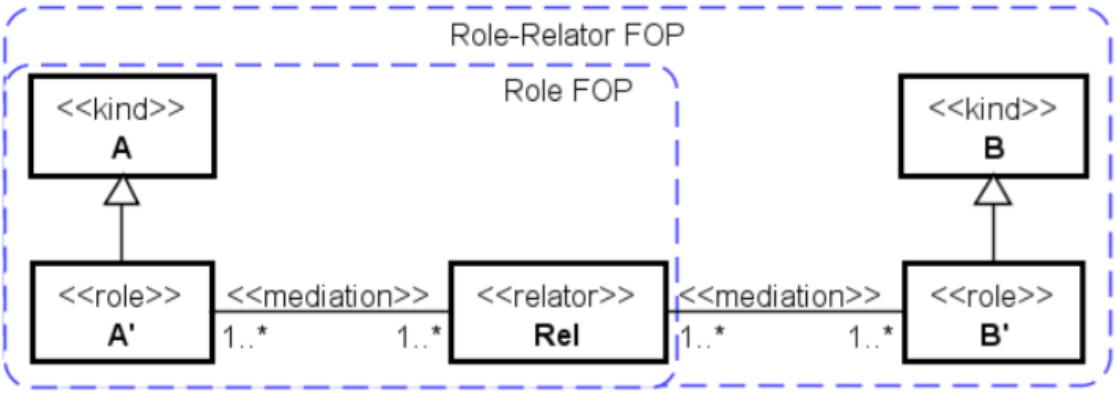

Figura 13 - Role-Relator FOP

O FOP apresentado acima demonstra a capacidade dos padrões serem aplicados de forma recursiva. Como visto no Role-Relator FOP, a criação de um pequeno fragmento de modelo resulta na necessidade de se aplicar outros padrões para manter o modelo completo. Entende-se por modelo completo, um modelo que esteja respondendo todas as regras de fundamentação da Ontologia de Fundamentação (i.e. um alinhamento entre o modelo e as regras de fundamentação). Nesse sentido, temos o porque de OntoUML ser vista como uma linguagem de padrões. Dito isso, temos que a maioria dos FOPs que serão apresentados na sequência não sejam FOPs completos, uma vez que estes prescrevem a necessidade de se especificar seus tipos genéricos com tipos específicos, como demonstrado anteriormente para o Relator FOP e Role FOP. Entretanto, dessa forma garantimos a formação de padrões genéricos e que possam servir para variados casos específicos. Para simplificar a especificação destes tipos específicos, apresentamos no próximo capítulo uma serie de procedimentos a serem usados visando a construção passo-a-passo de modelos completos e guiados por esses padrões. Na sequência apresentaremos a descrição do catálogo de padrões embutidos na linguagem OntoUML.

3.2 O CATÁLOGO DE PADRÕES ONTOUML

Nesta seção apresentamos os padrões já descobertos de OntoUML bem como os novos padrões identificados pela nossa análise. Este catálogo está estruturando com as seguintes informações:

Nome: o nome específico definido para o padrão;
Propósito: descreve a intenção do padrão e o problema no qual ele aborda;
Fragmento: fragmento de modelo mostrando o padrão e suas variações, bem como discussões sobre seus aspectos de modelagem;
Exemplo de uso: exemplo de aplicação do padrão;

Como já citado anteriormente, alguns padrões apresentam estereótipos abstratos que devem ser substituídos por algum de seus subtipos específicos. Nesse sentido, a Figura 1 deve ser usada para verificar os tipos na hierarquia de classes de UFO que são compatíveis com um determinado estereótipo abstrato. Esta decisão foi tomada para manter os padrões sob uma estrutura mais simples e resumida, deixando a critério do usuário definir quais estereótipos ele irá usar a cada aplicação do padrão. Porém, para simplificar a leitura podemos resumir os tipos abstratos com seus respectivos tipos específicos como listado abaixo:

Sortal: Kind, Quantity, Collective, Subkind, Phase ou Role;
Endurant: Qualquer um dos tipos instanciáveis de OntoUML;
Relational Anti-Rigid: Role ou RoleMixin;
Rigid Sortal: Kind, Quantity, Collective ou Subkind;
Rigid Type: Kind, Quantity, Collective, Subkind ou Category;
Anti-Rigid Type: Role, Phase, RoleMixin ou Mixin;
Moment: Relator ou Mode.

Nos diagramas que seguem os padrões foram apresentados, as cardinalidades das associações presentes no diagrama foram omitidas para tornar mais genérica a aplicação desses padrões, exceto quando fizer parte do padrão. Porém, para exemplos de uso aplicamos as cardinalidades a cada exemplo. Ademais, os nomes dados as classes em cada padrão servem apenas como referencia para sua explicação, não tendo nenhum efeito sobre a aplicação dos padrões.

Principle of Identity FOPs

Nome: Principle of Identity FOP.

Propósito: em UFO temos que todos os *Sortais* de um modelo necessariamente precisam ter ou prover um princípio de identidade para suas instâncias. Dessa forma, este padrão busca fornecer uma opção de modelagem para resolver esta necessidade.

Fragmento:

Figura 14 - Principle of Identity FOPs

Na figura acima temos as duas variações deste padrão. A esquerda temos uma especialização de um Sortal *Specific* para um Sortal *General*. Neste caso, temos que ambos os tipos precisarão ser substituídos por um tipo específico. Entretanto, ainda permanece a possibilidade da classe *General* não prover identidade para seus subtipos (e.g. Role *General*). Para estes casos, temos o fragmento a direita, onde em última instância a classe *General* pertencerá a um dos tipos provedores de identidade (i.e. *Substance Sortal*).

Exemplo de uso:

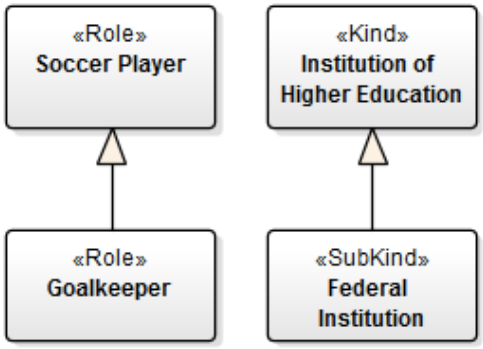

Figura 15 - Exemplo de uso: Principle of Identity FOP

Em ambos os exemplos temos a aplicação direta do Principle of Identity FOP. No exemplo a esquerda temos a aplicação do padrão sobre dois *Sortais* e no exemplo a direita a aplicação de um *Sortal* e um *Substance Sortal*.

Relator FOPs

Nome: Relator FOP.

Propósito: por definição, todo *Relator* precisa direta ou indiretamente estar relacionando via relações de mediação (*Mediation Relation*) com ao menos dois indivíduos distintos. Este padrão busca resumir estas propriedades em um fragmento único de modelo.

Fragmento:

Variação – Relator FOP

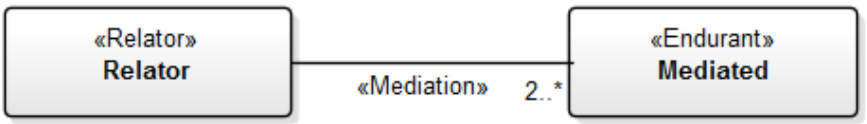

Figura 16 - Simple Relator FOP

A figura a cima apresenta a versão mais simples dos padrões para *Relators*. Como pode ser visto, esse padrão atende a premissa básica imposta por UFO para o uso deste tipo, i.e. uma instância de do tipo *Relator* deve conectar-se, via Mediation Relation, com ao menos dois indivíduos distintos. Nos padrões que se seguem, apresentamos variações deste padrão para os casos em que os tipos mediados pelo *Relator* possam possuir diferentes estereótipos.

Variação – Multiple Generic Relator FOP

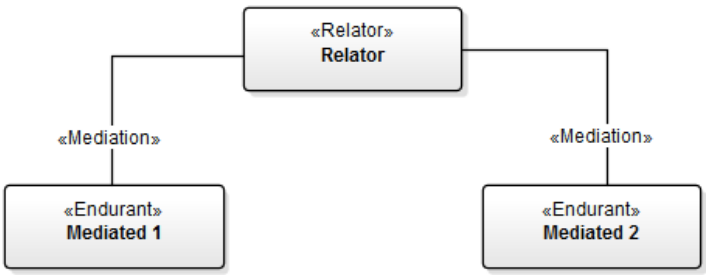

Figura 17 - Multiple Generic Relator FOP

Na figura acima temos uma das variações do Relator FOP. Nessa forma genérica temos a relação da classe *Relator* com dois *Endurants*. Nesse caso, os Endurants precisam ser substituídos por algum dos tipos instanciáveis de OntoUML.

Variação – Generic Relator FOP

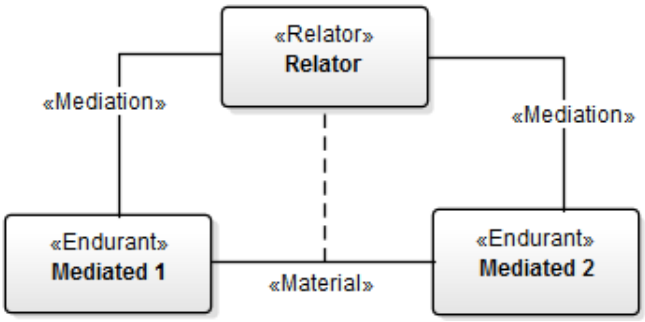

Figura 18 - Generic Relator FOP

Nessa segunda variação das regras de uso da classe *Relator* temos que as relações materiais entre os membros mediados pelo *Relator* (i.e. *Mediated 1* e *Mediated 2*) são derivadas via uma *Derivation Relation* do *Relator* para a *Material Relation*, o que resulta no fragmento da Figura 18 acima.

Variação – Multiple Generic Relator FOP

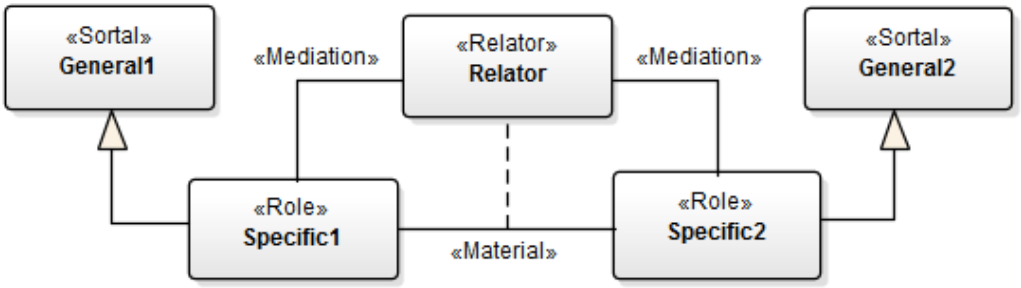

Figura 19 - General Relator FOP

Nesta última variação temos o caso mais comum de aplicação para *Relators*. Como discutido em (GUIZZARDI, 2005), tipicamente *Relators* aparecem mediando e derivando a relação material entre dois ou mais *Roles*, que por sua vez são subtipos de algum *Sortal*.

Exemplo de uso:

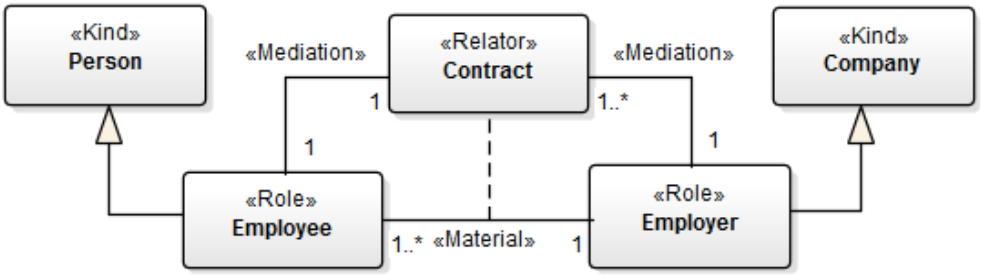

Figura 20 - Exemplo de Uso - General Relator FOP

A aplicação do Relator FOP é amplamente discutida na literatura sobre OntoUML e demonstram a importância deste padrão na modelagem conceitual utilizando esta linguagem. No exemplo da Figura 20 temos a aplicação clássica do padrão como mostrado primeiramente em (GUIZZARDI, 2005). A aplicação do General Relator FOP acima demonstra o relacionamento de contratação entre um empregado (*Employee*) e seu empregador (*Employer*). Como pode ser visto, o contrato entre as partes é reificado sob a figura do *Relator* entre eles, o qual agrupa todas as obrigações e deveres de ambas as partes nesse relacionamento.

RoleMixin Dependence FOP

Nome: Role Mixin Dependence FOP

Propósito: por definição, *RoleMixins* são tipos que agrupam ao menos dois tipos cujo as instâncias sejam anti-rígidas. Além disso, *RoleMixins* também são relacionalmente dependentes, o que implica na necessidade de relacioná-los. Este padrão visa demonstrar um fragmento de modelo capaz de responder essas regras.

Fragmento:

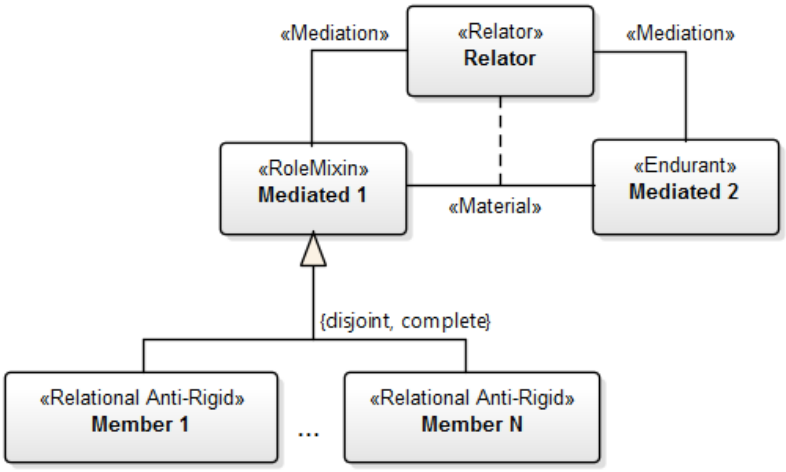

Figura 21 - RoleMixin Dependence FOP

Como pode ser visto acima, temos a generalização feita pelo *RoleMixin Mediated 1* para dois tipos *Relational Anti-Rigid*. Vale ressaltar que esta generalização necessariamente possuí ao menos dois subtipos. Outro ponto a ser destacado são as possibilidades disponíveis para os tipos *Relational Anti-Rigid*, como dito anteriormente é possível substituir este tipo por um *Role* ou outro *RoleMixin*. Nesses casos teremos novamente a necessidade de se aplicar outros padrões para manter o modelo completo.

O relacionamento do *RoleMixin Mediated 1* é similar ao explicado para o Generic Relator FOP. Logo, temos que o *Endurant Mediated 2* também precisa ser alterado para um dos tipos específicos da linguagem. Entretanto, caso utilizássemos o Relator FOP poderíamos ter *Mediated 1* idêntico a *Mediated 2*.

Exemplo de uso:

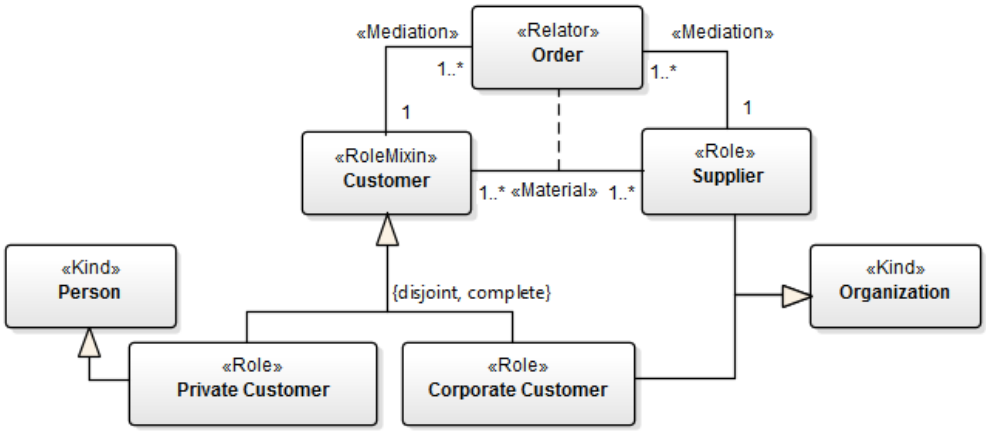

Figura 22 - Exemplo de Uso - RoleMixin Dependence FOP

Na figura acima temos a aplicação do RoleMixin Dependence FOP estendendo o clássico exemplo sobre *RoleMixins* apresentado em (GUIZZARDI, 2005). Aqui apresentamos a relação entre um cliente (*Customer*) e um fornecedor (*Supplier*). Neste exemplo temos que tanto uma organização (*Organization*) quanto uma pessoa (*Person*) podem ser clientes. Um ponto importante a se destacar neste exemplo é a aplicação concatenada do padrão *Principle of Identity FOP* para suprir o principio de identidade das classes estereotipadas como *Roles* presentes no modelo.

Partition FOPs

Nome: Partition FOP;

Propósito: na modelagem conceitual, temos que uma partição (*Partition*) é a divisão de um tipo em diferentes subtipos em decorrência de alguma propriedade importante o suficiente para gerar essa divisão. Como exemplo temos a propriedade gênero que particiona o tipo Pessoa em Homem e Mulher, ou a propriedade idade que particiona o tipo Pessoa em Criança, Adulto ou Idoso. Como pode ser visto, o ato de se particionar um tipo implica na necessidade de se instanciar obrigatoriamente um de seus subtipos. Este padrão propõe os fragmentos possíveis de particionamentos para OntoUML.

Fragmento:

Variação – Role Partition FOP

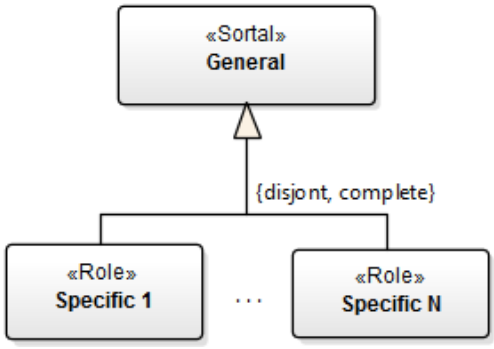

Figura 23 - Role Partition FOP

Variação – Phase Partition FOP

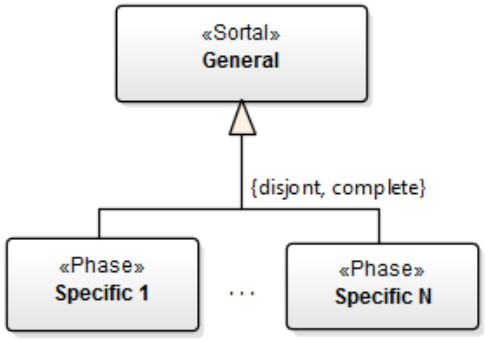

Figura 24 - Phase Partition FOP

Variação – Subkind Partition FOP

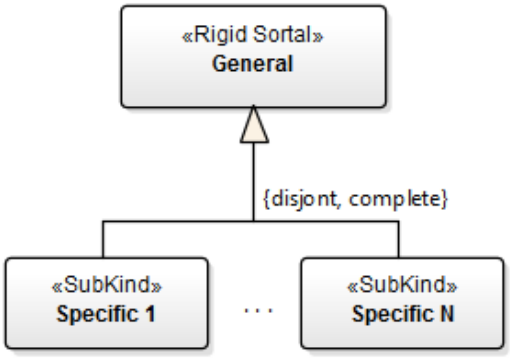

Figura 25 - Subkind Partition FOP

Variação – Category Partition FOP

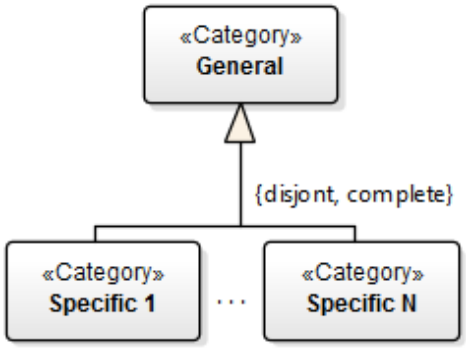

Figura 26 - Category Partition FOP

Variação – RoleMixin Partition FOP

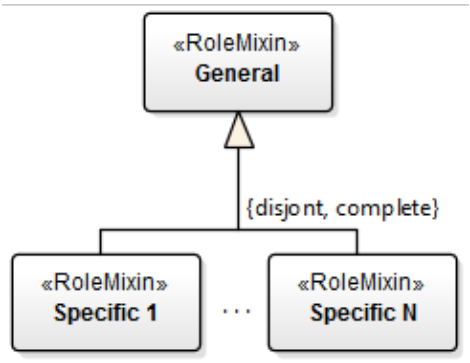

Figura 27 - RoleMixin Partition FOP

Variação – Mode Partition FOP

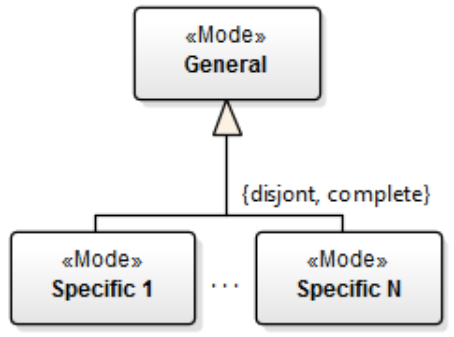

Figura 28 - Mode Partition FOP

Variação – Relator Partition FOP

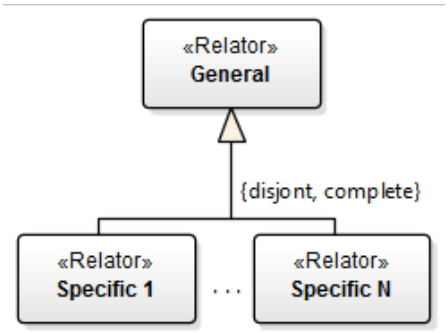

Figura 29 - Relator Partition FOP

Como pode ser visto, em cada uma das variações, existem vários tipos de partições em OntoUML, entretanto devido as regras de fundamentação definidas em UFO, estas possibilidades se limitam aos fragmentos apresentados aqui. É Importante notar que para cada variação temos necessariamente ao menos dois subtipos para cada tipo. Outro ponto importante é a necessidade de se configurar o Generalization Set (um conjunto de especializações que possuem o mesmo supertipo em comum, as quais podem ou não ter propriedades de disjunção ou completude associadas) de todas as partições como disjuntas e completas (disjoint e complete) o que de fato caracteriza a formação da partição, dessa forma garantido que para toda instancia do tipo *General*, ela estará instanciando também exatamente um de seus subtipos (*Specific*).

Exemplo de uso:

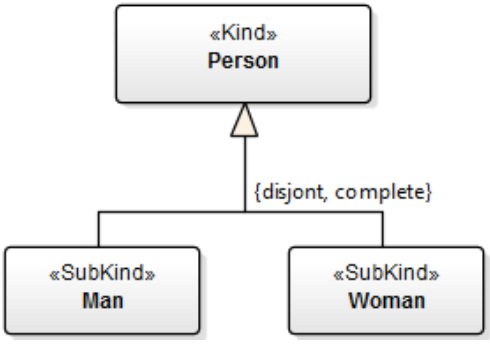

Figura 30 - Exemplo de Uso - Subkind Partition FOP

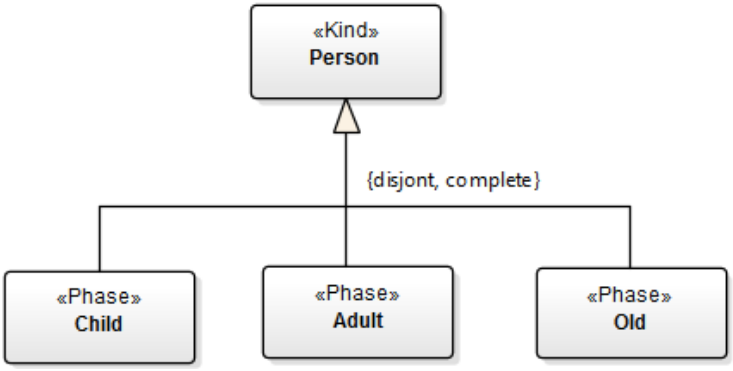

Figura 31 - Exemplo de Uso - Phase Partition FOP

Nos exemplos acima temos a aplicação do Subkind Partition FOP e Phase Partition FOP, respectivamente. Como discutido anteriormente, esses padrões são usados para particionar um tipo (e.g. *Person*) em diferentes subtipos. Em ambos os casos temos o Generalization Set configurado como disjunto e completo (disjoint e complete) o que força a necessidade das instâncias do supertipo (i.e. *Person*) pertencerem a exatamente uma de seus subtipos, caracterizando assim a existência da partição.

Generalization FOPs

Nome: Generalization FOP;

Propósito: na modelagem conceitual generalização de tipos ocorrem quando dois ou mais tipos distintos possuem alguma propriedade. Essa propriedade em comum pode ser então representada em um novo tipo que generaliza os tipos em questão. Isso também ocorre em OntoUML na qual as regras de fundamentação impostas por UFO limitam as possibilidades de relacionamento dos tipos (GUIZZARDI, 2005). Este padrão propões uma sequência de fragmentos demonstrando as possíveis generalizações entre os tipos de OntoUML.

Fragmento:

Variação – Category Generalization FOP

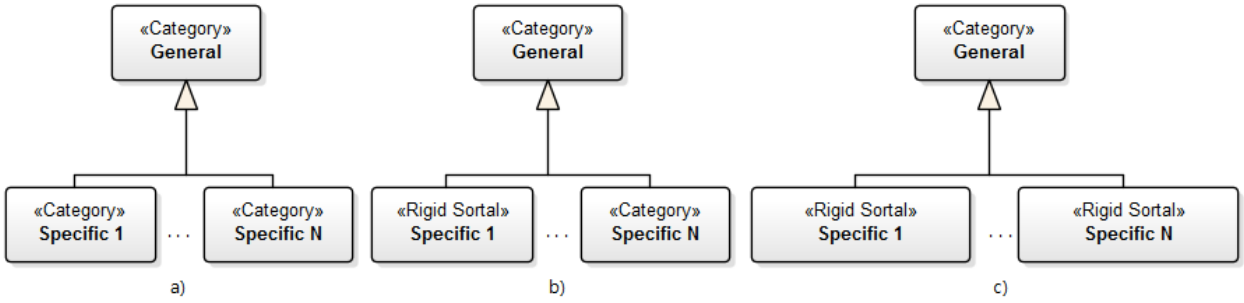

Figura 32 - Category Generalization FOP

No exemplo a cima temos que em *a)* é possível que o *Category General* possua N subtipos do tipo *Category*; em *b)* que o *Category General* pode generalizar o misto de tipos rígidos (*Rigid Sortal*) e de *Categories*; por fim em *c)* temos que o *Category General* também pode generalizar somente tipos rígidos (*Rigid Sortal*) também.

Variação – Mixin Generalization FOP

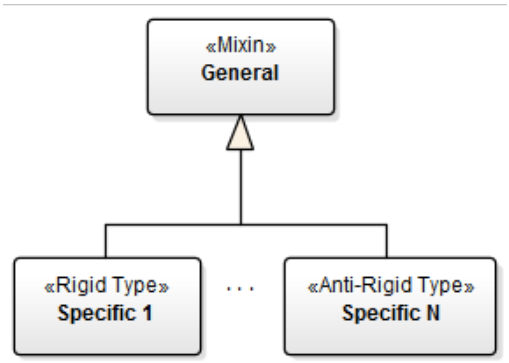

Figura 33 - Mixin Generalization FOP

Neste fragmento temos que o *Mixin General* pode generalizar um misto de tipos rígidos (*Rigid Type*) e de tipos anti-rígidos (*Anti-Rigid Type*) mas obrigatoriamente contando com ao menos um de cada um desses dois.

Variação – RoleMixin Generalization FOP

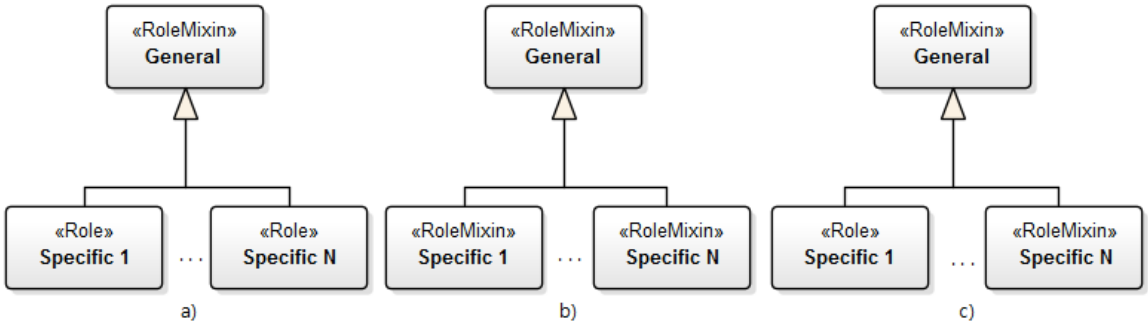

Figura 34 - RoleMixin Generalization FOP

No exemplo a cima temos que em *a)* é possível que o *RoleMixin General* possua N subtipos do tipo *Role*; em *b)* temos que o *RoleMixin General* também pode generalizar somente tipos *RoleMixin*; por fim em *c)* temos que o *RoleMixin General* pode generalizar o misto de tipos *Roles* e *RoleMixin.*

Nas variações apresentadas acima, temos as formas possíveis de se criar generalizações em OntoUML. Vale destacar que diferente dos Partition FOPs, os Generalization FOPs não precisam ser disjuntos nem completos, porém como no caso das partições, precisam ter ao menos dois subtipos para classe *General*.

Exemplo de uso:

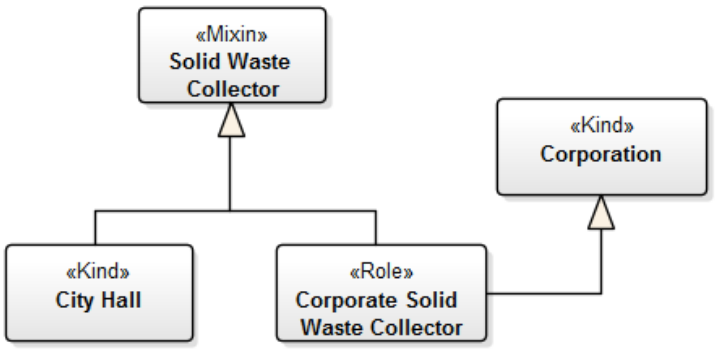

Figura 35 - Exemplo de Uso - Mixin Generalization FOP

No exemplo acima temos modelado a generalização da propriedade de ser um coletor de resíduos sólidos (*Solid Waste Collector*). De acordo com as leis brasileiras, temos que tanto uma prefeitura (*City Hall*) quanto uma organização (*Corporate Solid Waste Collector*) prestam este serviço, entretanto apenas para as organizações esta propriedade é opcional.

Dessa forma, criamos um supertipo *Mixin* o qual caracteriza o generalização (agrupamento) dos tipos rígidos e anti-rígidos que possuem esta propriedade.

Relation FOPs

Nome: Relation FOP;

Propósito: em OntoUML temos diferentes tipos de relações bem como regras impostas por UFO que definem os possíveis tipos a serem relacionados. Este padrão propõe uma lista de fragmentos que simplificam quais são as possíveis configurações para essas relações.

Fragmento:

Variação – Formal FOP

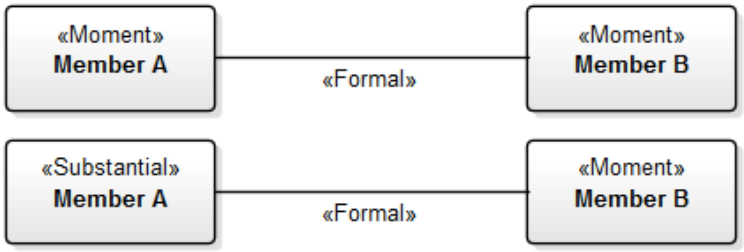

Figura 36 - Formal FOP

Variação – Characterization FOP

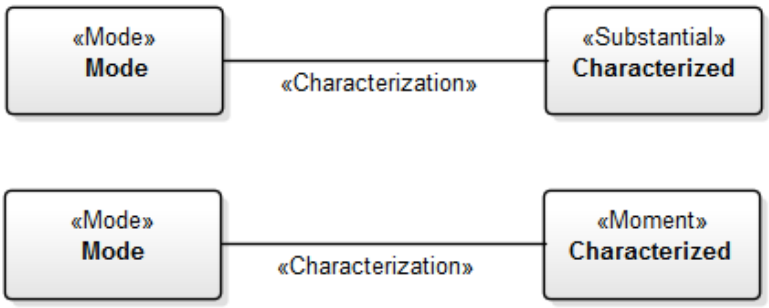

Figura 37 - Characterization FOP

Como demonstrado nas figuras acima, temos que cada padrão de relacionamento possuí suas próprias regras de configuração. Entretanto, para os casos das relações meronímicas (relações de parte-todo) os fragmentos válidos possíveis são mais complexos de serem gerados, uma vez que as regras de aplicação destas relações levam em consideração os tipos, subtipos e

supertipos dos membros da relação. Diante disto, demonstramos abaixo o padrão para esses casos e as regras de formação para cada uma das relações meronímicas de OntoUML.

Exemplo de uso:

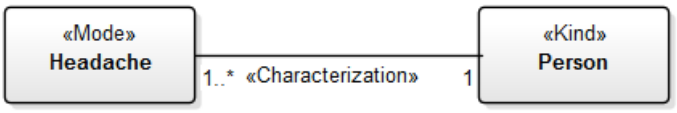

Figura 38 - Exemplo de Uso - Characterization FOP

No exemplo acima temos a aplicação do Characterization FOP, onde temos uma relação de *Characterization* relacionado uma pessoa (*Person*) e um Mode dor de cabeça (*Headache*).

Meronymic FOP

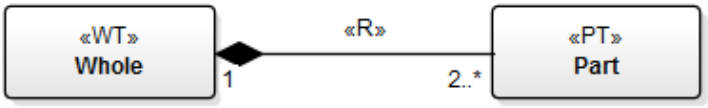

Figura 39 - Generic Meronymic Form

O fragmento acima é uma forma genérica utilizada aqui para facilitar o entendimento deste padrão. O estereótipo R da relação deve ser substituído por algum estereótipo de relação meronímica de OntoUML como: *ComponentOf*, *SubQuantityOf*, *SubCollectionOf* e *Member Of*. Utilizaremos também a variável *T* que pode ser um Role, Phase ou Subkind de forma excludente, ou seja, apenas um dos três. A cardinalidade mínima do lado da parte (2) é definida para todas as relações meronímicas quando o tipo-todo se relaciona apenas com apenas um tipo-parte pela mesma relação meronímica R (GUIZZARDI, 2005). Ademais, os estereótipos WT e PT serão definidos em cada uma das relações abaixo.

Variação – ComponentOf FOP

R = ComponentOf;

PT ou **WT** = Kind;

PT ou **WT** = Qualquer *T* que tenha um Kind como supertipo mais alto da hierarquia (i.e. qualquer functional complex universal);

PT ou **WT** = Qualquer Mixin Universal cujo subtipos sejam Kinds ou *T*s que possuam um Kind como supertipo mais alto da hierarquia (i.e. qualquer functional complex universal).

Variação – SubQuantityOf FOP

R = SubQuantityOf;
PT ou **WT** = Quantity;
PT ou **WT** = Qualquer *T* que tenha um Quantity como supertipo mais alto da hierarquia;
PT ou **WT** = Qualquer Mixin Universal cujo subtipos sejam Quantities ou *T*s que possuam um Quantity como supertipo mais alto da hierarquia.

Variação – SubCollectionOf FOP

R = SubCollectionOf;
PT ou **WT** = Collective;
PT ou **WT** = Qualquer *T* que tenha um Collective como supertipo mais alto da hierarquia;
PT ou **WT** = Qualquer Mixin Universal cujo subtipos sejam Collectives ou *T*s que possuam um Collective como supertipo mais alto da hierarquia.

Variação – MemberOf FOP

R = MemberOf;
WT = Collective;
WT = Qualquer *T* que tenha um Collective como supertipo mais alto da hierarquia;
WT = Qualquer Mixin Universal cujo subtipos sejam Collectives ou *T*s que possuam um Collective como supertipo mais alto da hierarquia.
PT = Collective ou Kind;
PT = Qualquer *T* que tenha um Collective ou um Kind como supertipo mais alto da hierarquia;
PT = Qualquer Mixin Universal cujo subtipos sejam Collectives, Kinds ou *T*s que possuam um Collective ou Kind como supertipo mais alto da hierarquia.

Como mostrado acima, as configurações possíveis para relações meronímicas são mais complexas do que os demais padrões. Para simplificar o uso desses padrões, os fluxos apresentados no próximo capítulo são indicados para os modeladores novatos de OntoUML.

Exemplo de uso:

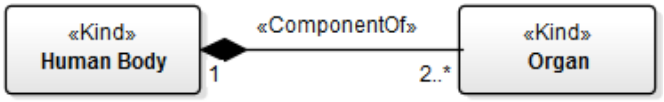

Figura 40 - Exemplo de Uso - Component Of FOP

Neste segundo exemplo demonstramos a aplicação do Component Of FOP. Como pode ser visto, temos que um corpo humano (*Human Body*) é composto de dois ou mais órgãos (*Organ*).

3.3 TRABALHOS RELACIONADOS

Analisando a literatura sobre padrões ontológicos e padrões de OntoUML encontramos os trabalhos (FALBO et al., 2013) e (GUIZZARDI; GRAÇAS; GUIZZARDI, 2011), os quais relacionaremos com o nosso trabalho.

Em (FALBO et al., 2013) é apresentado uma classificação dos padrões ontológicos existentes bem como um catálogo online colaborativo para padrões ontológicos. Distinguimos desse trabalho ao restringirmos nosso estudo aos padrões de fundamentação ontológicos (FOPs). Uma vez que nosso objetivo era identificar padrões de UFO, uma ontologia de fundamentação, optamos por restringir nosso escopo de estudo a esta classificação de padrões. Nesse aspecto, nosso catálogo de padrões é apresentado sob uma análise das regras de fundamentação da linguagem, não havendo possibilidades de identificação coletivas de padrões, ou sejam, uma vez que as regras de fundamentação são independentes de domínio, não há possibilidade de identificação de novas regras. Entretanto, como discutido em (RUY et al., 2011), técnicas para construção de bibliotecas de padrões de domínio (DROPs) para OntoUML também tem sido estudadas apresentando resultados positivos e possibilitado a aplicação dos padrões apresentados aqui como insumo para trabalhos futuros visando o uso combinado de padrões ontológicos (FOPs e DROPs) e possibilitando uma futura base de dados coletivas para DROPs.

Por outro lado, o trabalho apresentado em (GUIZZARDI; GRAÇAS; GUIZZARDI, 2011) se assemelha ao nosso trabalho ao lidar com padrões voltados para OntoUML. Entretanto, diferente da abordagem proposta pelos autores, onde apenas as regras para os tipos Kind, Subkind, Phase e Role (excluindo-se as regras de dependência) foram analisadas, em nosso trabalho analisamos todos os tipos de OntoUML. Outro ponto que diferencia os dois trabalhos é a implementação dos padrões. Enquanto em (GUIZZARDI; GRAÇAS; GUIZZARDI, 2011) um novo editor foi proposto para lidar apenas com os aspectos de uso dos padrões, nós implementamos nosso catálogo de padrões no OLED, possibilitando aos usuários o uso de todas as ferramentas desenvolvidas para modelagem utilizando OntoUML.

3.4 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Modelagem conceitual é uma tarefa difícil e complexa, porém acreditamos que os padrões apresentados aqui sirvam de guia para auxiliar a construções de modelos conceituais utilizando OntoUML. Entretanto ainda que tendo o catálogo apresentado aqui como guia, estima-se que novos usuários da linguagem tenham dificuldade de compreender quando e qual padrão aplicar durante o processo de modelagem. Diante disso, analisamos o processo de construção dos padrões buscando definir procedimentos de construção dos padrões para serem usados por usuários novatos da linguagem. Similar ao realizado em (GUIZZARDI; GRAÇAS; GUIZZARDI, 2011), verificamos que a partir da aplicação de um padrão de OntoUML, uma serie de outros padrões se fazem necessários para completar o modelo, gerando assim uma cadeia de chamadas para novos padrões, o que pretende ser analisado no próximo capítulo.

CAPÍTULO 4. PROCEDIMENTO PARA CONSTRUÇÃO DE MODELOS ONTOUML

4.1 UM PROCESSO GERAL PARA A CONSTRUÇÃO DE MODELOS ONTOUML

O processo de construção de modelos OntoUML é guiado por eventos de modelagem (ODELL, 1998). Estes eventos lidam com os aspectos da ontologia de fundamentação. Por exemplo, quando um modelador adiciona um novo tipo ao modelo, ele tem que definir sua rigidez, sua identidade (se própria ou herdada), sua dependência e as demais meta-propriedades da linguagem. Mesmo aplicando os padrões tratados no capítulo anterior, o modelador ainda necessita realizar uma serie de raciocínios para definir quando utilizar um padrão ou para verificar a completude do modelo. Nesse sentido, visando prover um método para garantir a consistência do modelo frente a todas essas decisões, nós defendemos a aplicação de procedimentos pré-definidos para construção de modelos OntoUML. Esses procedimentos visam responder as questões de meta-propriedade de OntoUML herdadas de UFO. Como já dito anteriormente, estas meta-propriedades representam as regras de fundamentação (i.e. as restrições impostas pela ontologia de fundamentação) e necessariamente precisam estar sendo atendidas pelos modelos.

Nesse sentido, a completude do modelo (i.e. a adequação do modelo as regras de fundamentação) depende dele estar respondendo todas as regras de fundamentação subjacente a linguagem na qual aquele modelo foi construído. Os procedimentos apresentados aqui visam suprir essa necessidade guiando os modeladores a criarem seus modelos de forma correta e completa. Para ilustrar isso, a Figura 41 abaixo demonstra o evento de modelagem para criação de um novo elemento em um modelo conceitual.

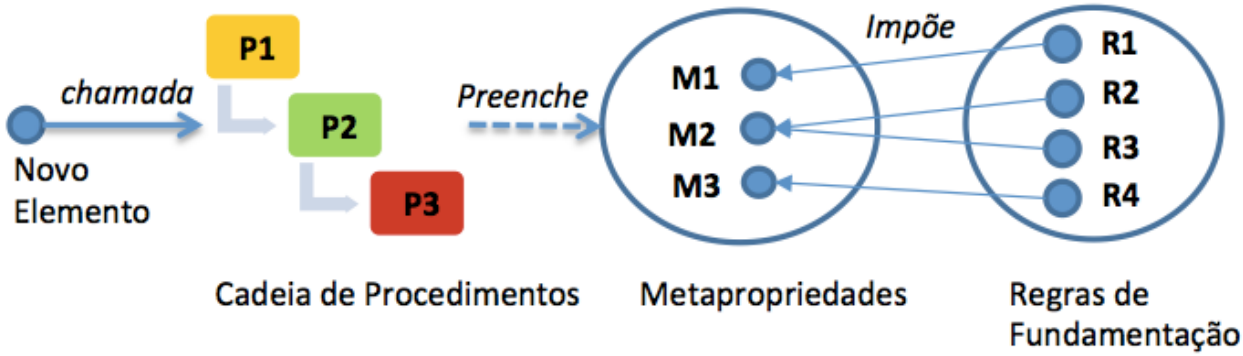

Figura 41 - Evento de Modelagem - Novo Elemento

Os procedimentos apresentados aqui foram extraídos dos padrões presentes em nosso catálogo de padrões como será discutido mais adiante. Entretanto, a diferença entre os padrões e os procedimentos é a capacidade desses de serem chamados de forma encadeada. Como demonstrado na figura acima, a adição de um novo elemento ao modelo força a chamada de um procedimento que por sua vez pode originar uma cadeia de chamadas a outros procedimentos, os quais pretendem preencher (atribuir valores) às meta-propriedades impostas pelas regras de fundamentação subjacentes a linguagem usada. Por exemplo, imaginemos a adição de uma classe *Student* estereotipada como *Role*. Esse novo elemento chamaria o procedimento para criação de *Roles*, o que por sua vez, verificaria a necessidade de se informar o tipo provedor de identidade do *Role*, uma vez que *Roles* não são provedores de identidade, mas apenas portadores. Nesse ponto, existe a identificação de que é necessário aplicar algum procedimento para suprir a identidade do *Role*, dessa forma uma chamada a um novo procedimento (ou a seleção de um elemento do modelo) deverá ser feita. Após isso, teremos o *Role Student* com sua identidade devidamente provida. Mas ainda assim existiria a necessidade de se criar a relação de mediação desse *Role*, dessa forma originando na chamada a novos procedimentos. Por fim, teremos realizado uma cadeia de chamadas de procedimentos os quais terão atendido a todas as regras de fundamentação impostas pela ontologia de fundamentação.

O procedimento genérico apresentado na Figura 41 resume a ideia dos procedimentos apresentados neste capítulo. Como demonstrado acima, a ideia geral é prover uma sequência de passos para auxiliar na construção de modelos OntoUML. Entretanto, embora os padrões do catálogo aqui proposto tendem a ser usados por usuários mais experientes da linguagem, os procedimentos aqui apresentados são formas objetivas e simples de se modelar, o que pretende ser uma abordagem voltada a auxiliar modeladores novatos da linguagem.

Diante disso, temos que o uso misto dos padrões e dos procedimentos tendem a ser uma abordagem mais completa para modelagem. Por exemplo, aplicando-se um *Generic Relator FOP* (Figura 18) podemos criar *Roles* sem seus supertipos deixando-os sem seus respectivos princípios de identidade, resultando em um modelo incompleto. Para solucionar isso, o procedimento de criação de *Roles* pode ser chamado para solucionar esta incompletude, verificando as necessidades do modelo e aplicando as mudanças necessárias. Diante disso, uma automatização desse processo é apresentado no fim deste capítulo como uma abordagem que utiliza em conjunto os padrões e os procedimentos para construção de modelos OntoUML.

Um ponto a ser destacado é que não estamos propondo um método (metodologia) para construção de modelos OntoUML, uma vez que este tipo de estudo demandaria uma análise quanto aos conceitos sendo modelados. Ao invés disso, os procedimentos apresentados aqui partem de outra premissa, ou seja, nós supomos que o conceito a ser modelado está de acordo com o domínio proposto, não fazendo parte dos nossos objetivos discutir se ele está correto ou não, dessa forma, apenas nos interessando o tipo OntoUML do conceito e suas propriedades.

O texto que se segue utiliza a classificação de tipos apresentada na Figura 42, onde é apresentado os tipos gerais dos procedimentos. Como pode ser visto nessa figura, existe uma distinção entre os Procedimentos Básicos e os Procedimentos Dependentes. Enquanto os procedimentos básicos lidam com novos elementos adicionados ao modelo, os procedimentos dependentes requerem o uso de mais de um tipo para sua execução, os quais são divididos entre procedimentos de Partição, Generalização e Relação.

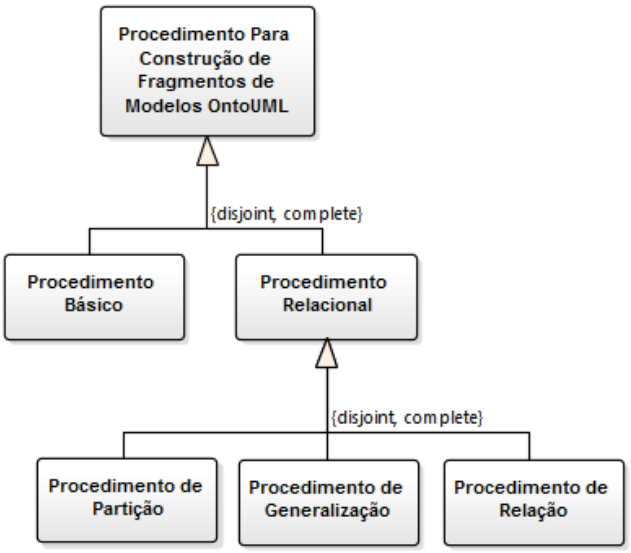

Figura 42 - Classificação geral dos procedimentos

Por fim, apresentamos em sequência os procedimentos para construção de fragmentos de modelo OntoUML. Esses procedimentos são aqui definidos por meio de diagramas de atividade de UML.

4.2 PROCEDIMENTOS BÁSICOS

4.2.1 Substance Sortal

Substance Sortals são tipos que representam as propriedades gerais de todas as entidades e são responsáveis por suprir o princípio de identidade para suas instâncias. O procedimento apresentado abaixo demonstra a forma de criação desses tipo.

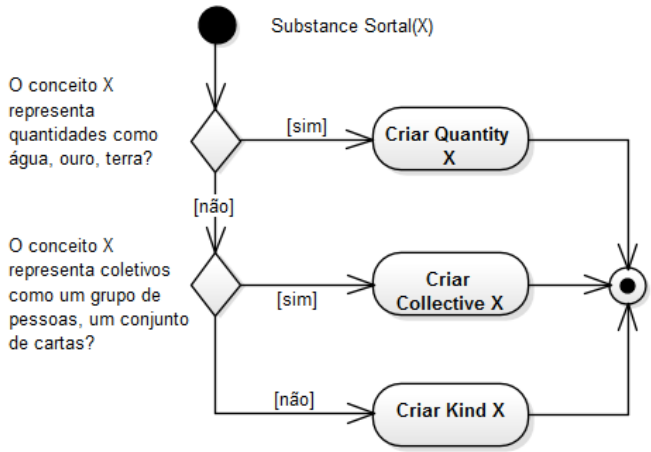

Figura 43 - Procedimento: Substance Sortal

Como pode ser visto na figura, dado um *Substance Sortal X*, o procedimento guia o usuário a criar o tipo específico de *X*.

4.2.2 Subkind

Subkinds são tipos rígidos e relacionalmente independentes que carregam o princípio de identidade fornecido por algum *Substance Sortal*. O procedimento abaixo demonstra o processo de criação de um conceito *Subkind X*. No geral, *Subkinds* funcionam como adição de informação à definição mais geral de um *Substance Sortal*. Por exemplo, temos os *Subkinds Man* e *Woman* do *Kind Person*. Neste caso, temos que ambos os *Subkinds* estão adicionando a propriedade de gênero ao *Kind Person*.

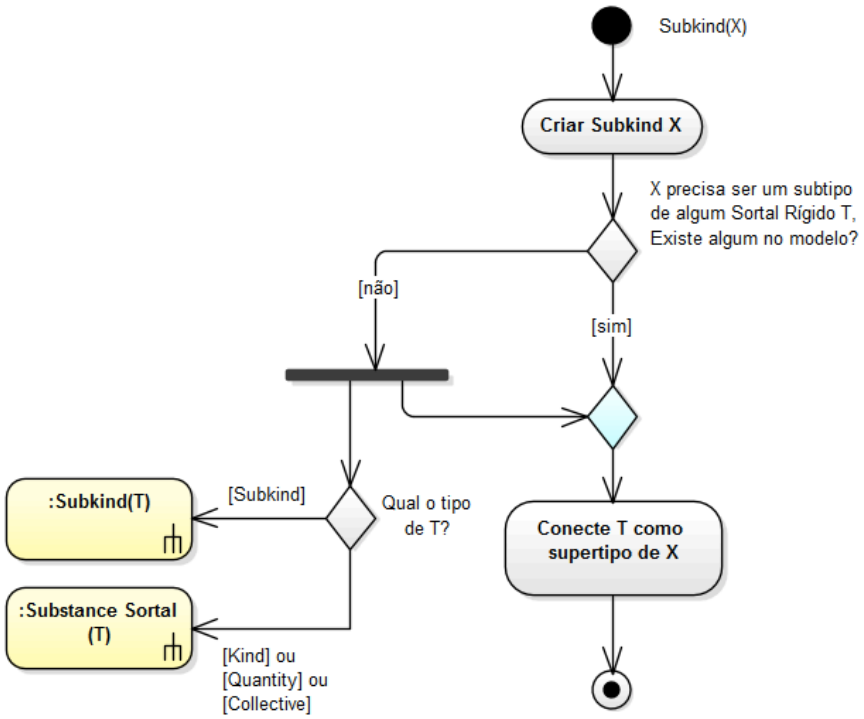

Figura 44 - Procedimento: Subkind

É importante notar que as chamadas aos novos procedimentos (em amarelo) recebe como parâmetro o tipo *T* a ser criado. Ademais, essas chamadas a outros procedimentos fundamenta a abordagem defendida neste trabalho quanto ao encadeamento dos procedimentos e padrões.

4.2.3 Role

Roles são tipos anti-rígidos e relacionalmente dependentes, os quais representam os papéis dos tipos envolvidos em um relacionamento. A figura abaixo formaliza o procedimento de criação de *Roles*.

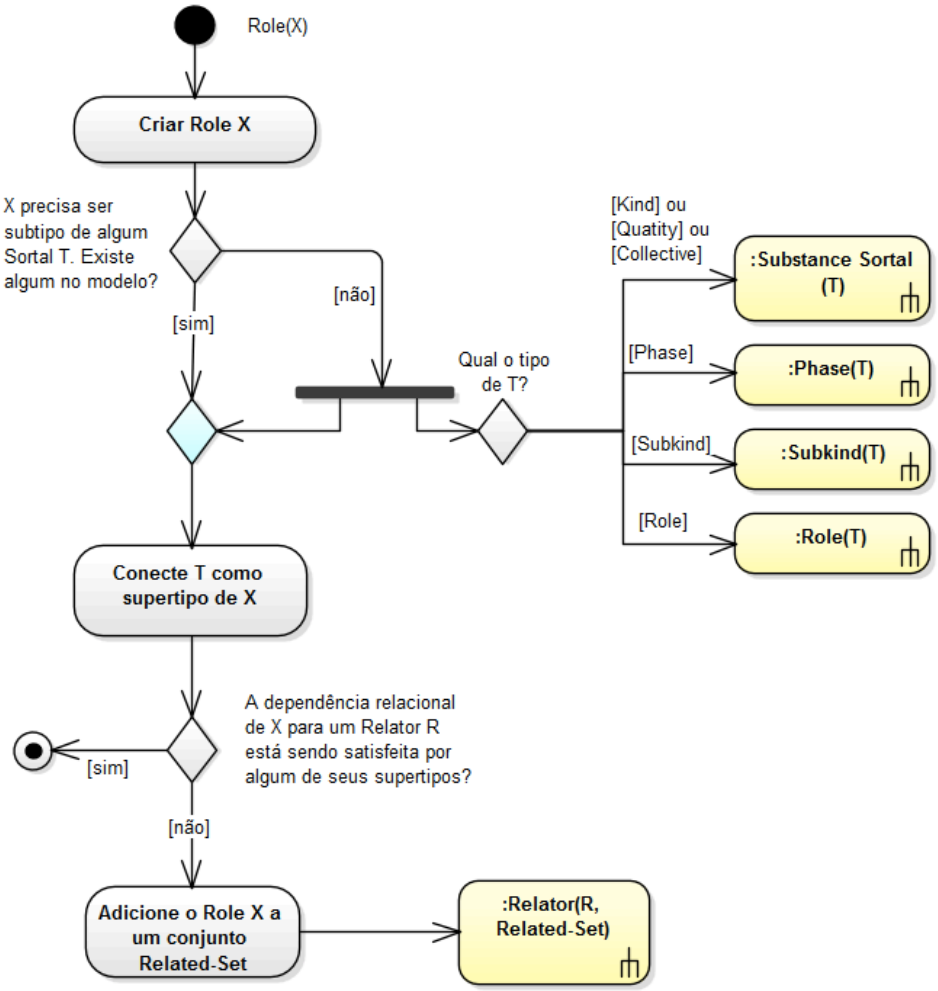

Figura 45 - Procedimento: Role

Como *Roles* não são provedores de identidade eles precisam estar conectado a algum *Sortal* como supertipo. Uma vez que os procedimentos de criação para todos os *Sortais* os obrigam a possuir um *Substance Sortal* como supertipo, esta chamada garante que em algum momento este *Role* possuirá um *Substance Sortal* como supertipo mais alto. Além disso, devido a dependência relacional dos *Roles* eles precisam estar conectados direto ou indiretamente a algum *Relator*. Para isso, o procedimento define a criação de um conjunto *Related-Set* que servirá como parâmetro para a chamada ao procedimento de criação de *Relator*. Esse conjunto representa os elementos que serão relacionados a este novo *Relator*, o qual será tratado no seu respectivo procedimento.

4.2.4 Phase

Em OntoUML, *Phases* representam tipos anti-rígidos e relacionalmente independentes. Como já discutido anteriormente, *Phases* necessariamente aparecem em partições de fases

(Phase Partitions) no modelo. Na figura abaixo temos definido o seu procedimento de criação.

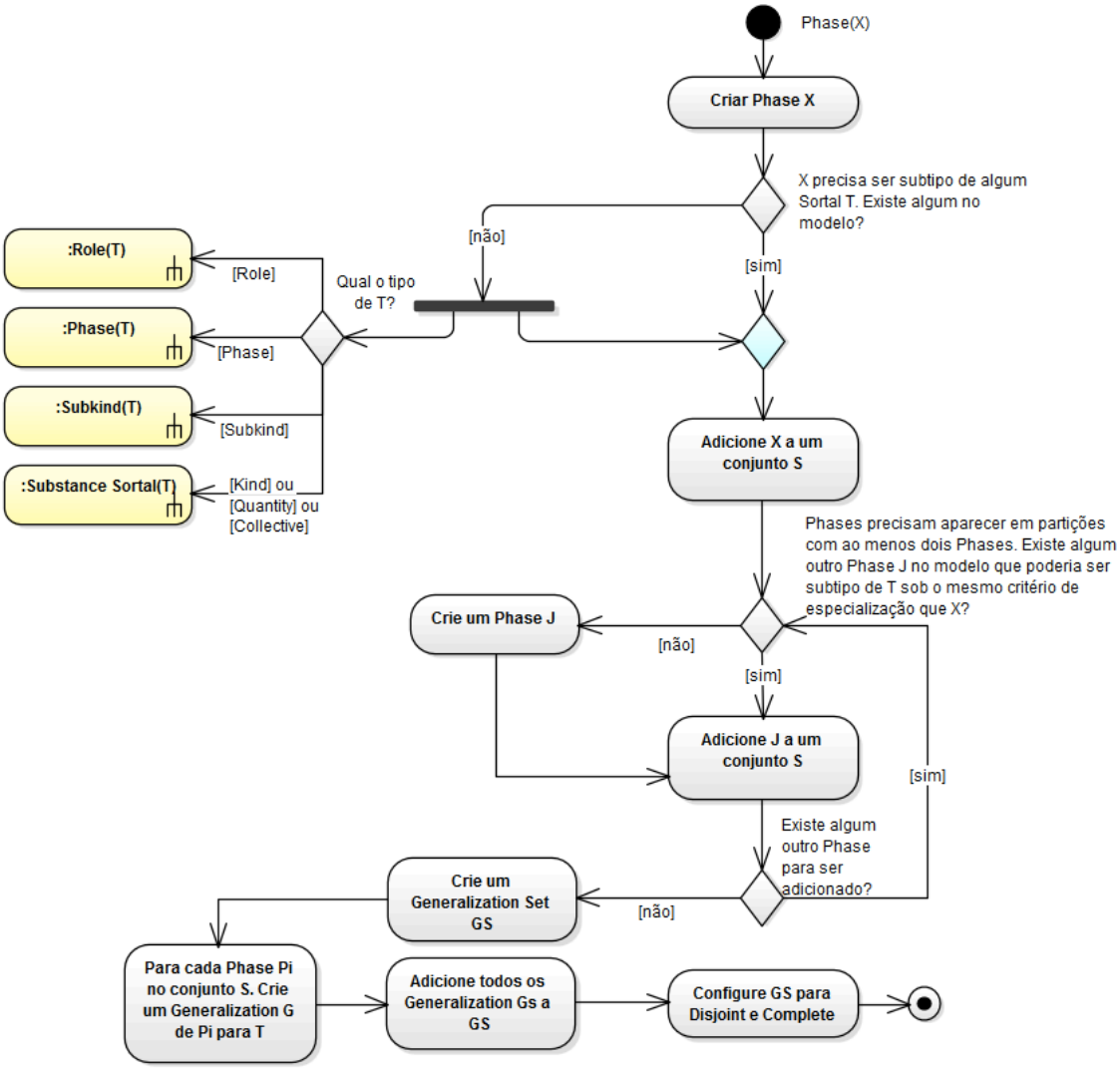

Figura 46 - Procedimento: Phase

Primeiramente, é identificado a necessidade de se criar um supertipo para a *Phase X*. Se for esse o caso (ou seja, caso não exista um candidato no modelo), o procedimento executa a criação de algum *Sortal* (que pode, portanto, ser um role, um phase, um subkind ou um substance sortal, como ilustrado no modelo). Após isso, o usuário é forçado a criar ou selecionar no modelo todos as possíveis fases completares ao *Phase X*, segundo o mesmo critério de partição, ou seja, todas as outras fases do mesmo phase partition. Em seguida, é criado um Generalization Set de todos essas fases, e por último as propriedades de disjunção e completude desse Generalization Set são configuradas.

4.2.5 Relator

Relator são entidades com a propriedade de conectar entidades diferentes. Nesse sentido, o procedimento descrito abaixo formaliza a criação de um tipo *Relator* e seus relacionamentos, na sequência, temos as regras de derivação das cardinalidades de cada relação criada pelo procedimento.

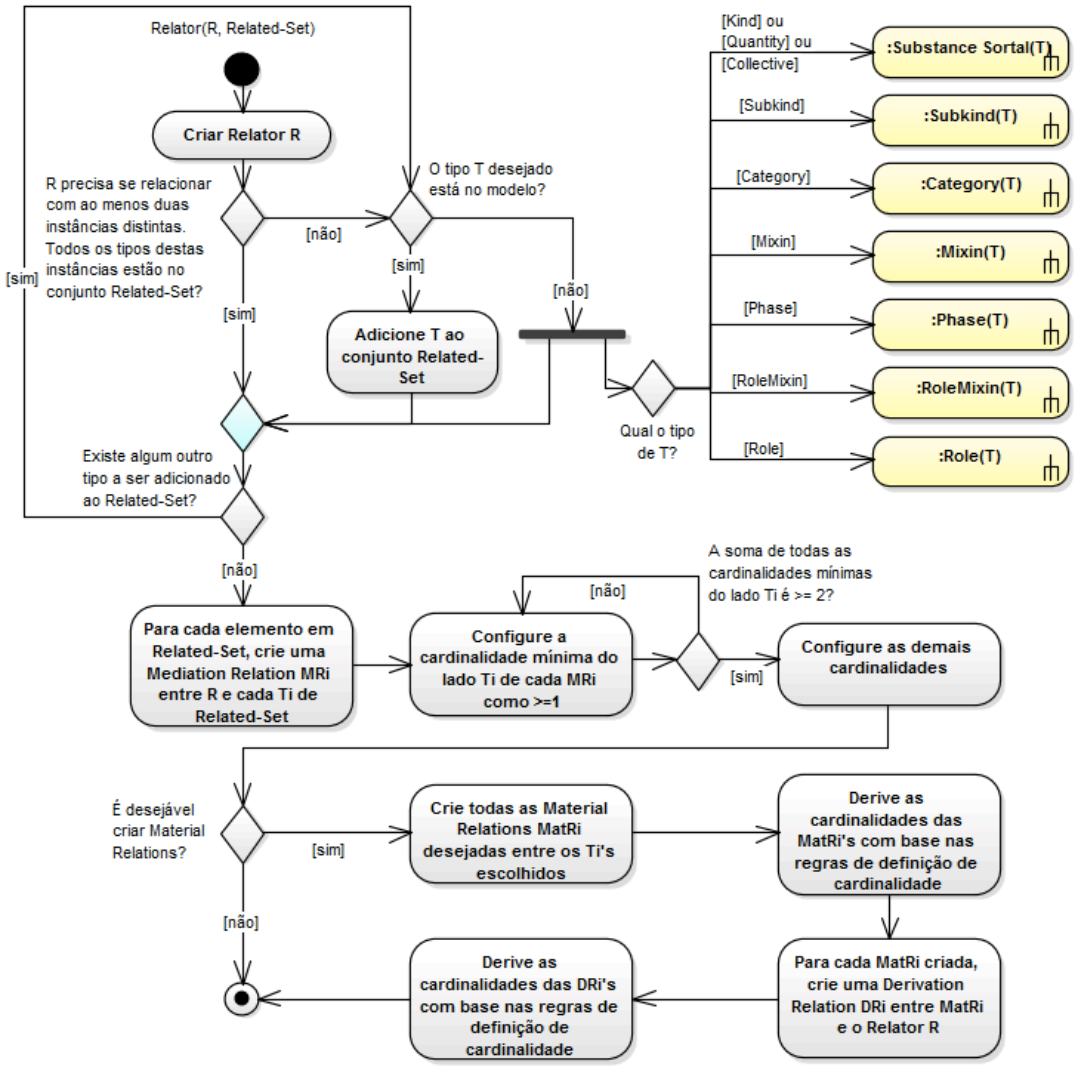

Figura 47 - Procedimento: Relator

Como demonstrado no procedimento acima, *Relators* podem conectar diferentes tipos de OntoUML. Entretanto, como argumentado em (GUIZZARDI, 2005) o mais indicado para se usar seriam *Roles* ou *RoleMixins*. Guizzardi defende isso ao argumentar que ao se usar *Roles* e *RoleMixins*, o modelo tende a ficar semanticamente mais rico, mais expressivo e pragmaticamente mais eficiente (BODART; PATEL; SIM et al., 2001), uma vez que os papéis desempenhado por cada membro do relacionamento estará explicitado.

Para cada par de entidades mediadas por um *Relator*, é possível existir uma relação *Material* entre elas. Entretanto, como argumentado em (GUIZZARDI, 2005) as cardinalidades da relação *Material* são derivadas das cardinalidades das relações de mediação, logo, esta relação é opcionalmente modelada. Para formalização das regras de definição das cardinalidades utilizamos a figura a baixo como exemplo.

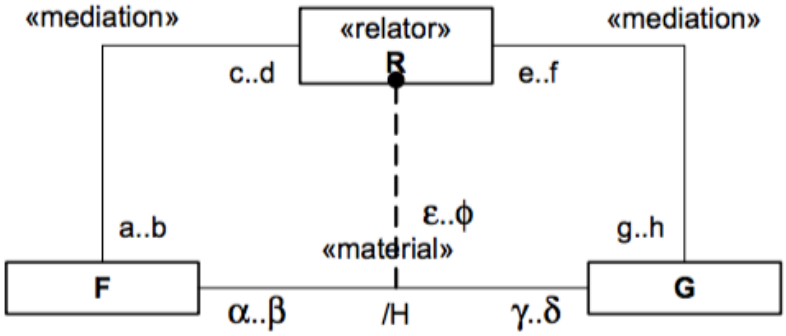

Figura 48 - Definição das cardinalidades para relacionamentos com Relators (GUIZZARDI, 2005)

Como definido em (GUIZZARDI, 2005), temos:

- α = e X a
- β = f X b
- γ = c X g
- δ = d X h
- ε = a X g
- ϕ = b X h

4.2.6 RoleMixin

RoleMixins são tipos que agrupam tipos anti-rígidos e com diferentes princípios de identidade. Assim como *Roles*, os *RoleMixins* também são relacionalmente dependentes, o que significa que eles precisam estar relacionados diretamente ou indiretamente (por algum de seus supertipos) com algum *Relator*. A Figura 49 abaixo apresenta o procedimento para criação de *RoleMixins*.

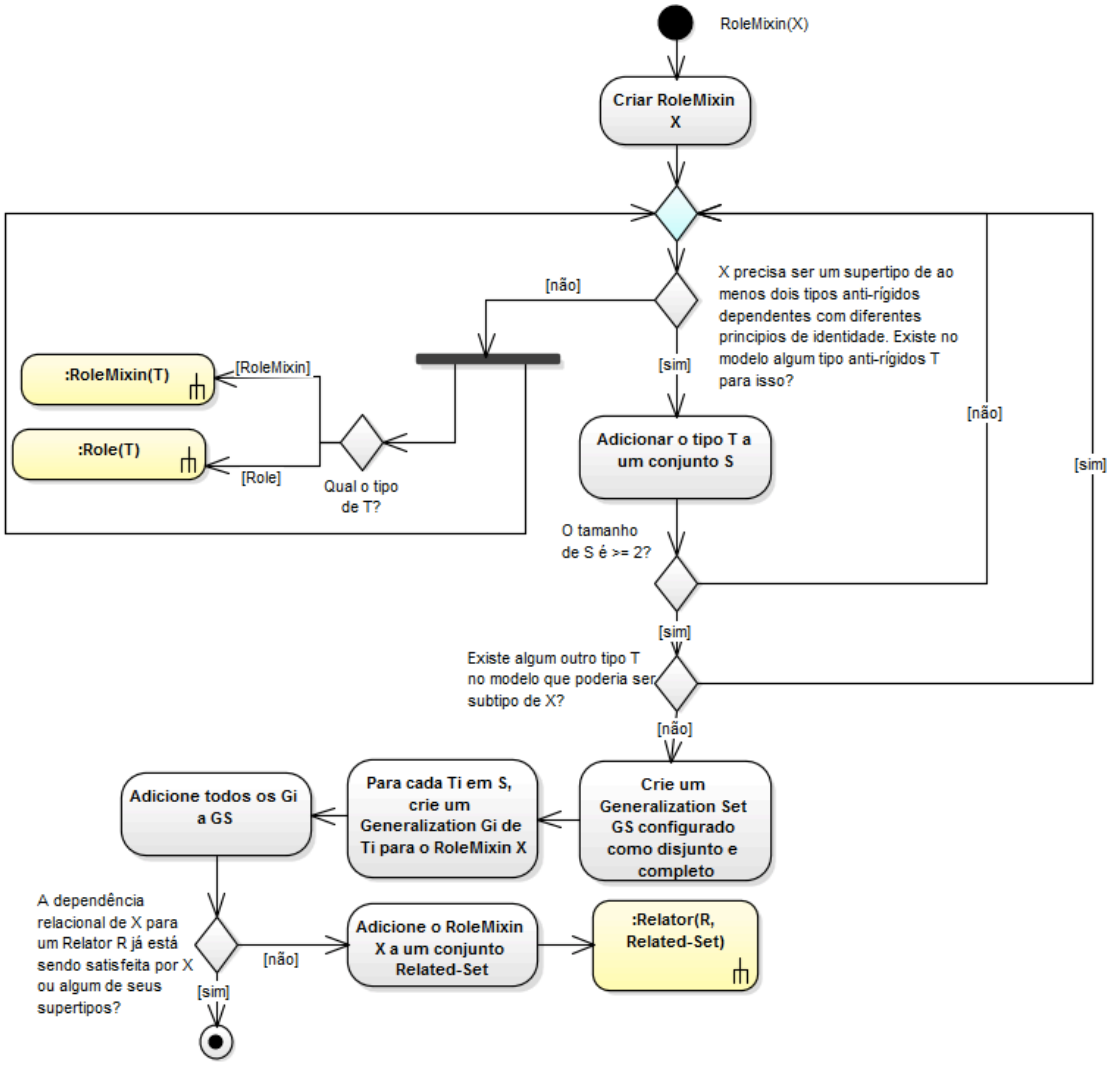

Figura 49 - Procedimento: RoleMixin

Como pode ser visto, o procedimento força o usuário a criar ou selecionar do modelo ao menos dois tipos anti-rígidos com diferentes princípios de identidade. Em sequência, criamos um Generalization Set disjunto e completo entre eles e verificamos se a dependência relacional do *RoleMixin* está sendo satisfeita, em caso negativo, chama-se o procedimento de criação de *Relators* para satisfazer esta regra tendo adicionado o *RoleMixin X* ao conjunto *Related-Set*.

4.2.7 Category

Categories representam tipos rígidos e relacionalmente independentes. No geral, eles agregam propriedades que são comuns a diferentes *Substance Sortals*. Na Figura 50 abaixo temos descrito o procedimento de criação do tipo *Category*.

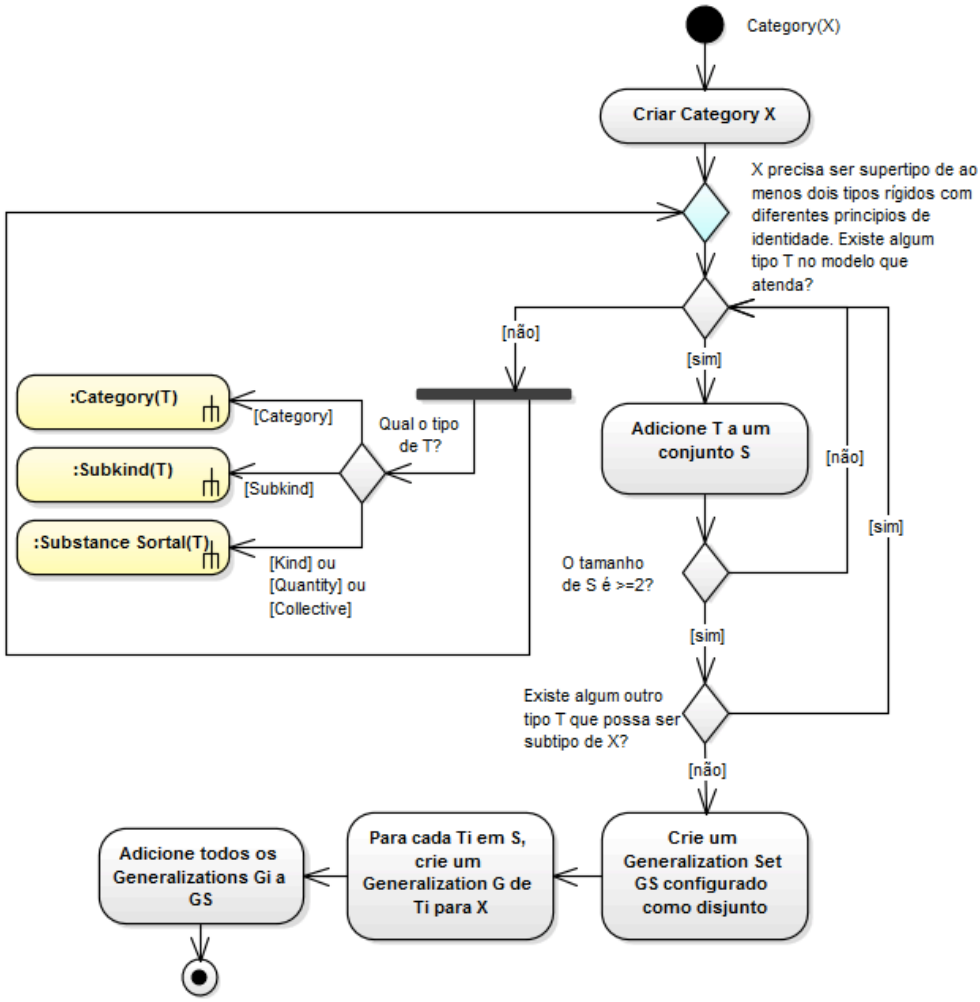

Figura 50 - Procedimento: Category

Primeiramente é indicado a necessidade de se escolher ao menos dois tipos rígidos no modelo que são portadores de diferentes princípios de identidade. Em seguida, esses são adicionados a um conjunto *S*, o qual será posteriormente usado para criação do Generalization Set entre o *Category* criado e seus subtipos. No caso de haver necessidade de se criar um novo tipo, é indicado a criação dos tipos rígidos de OntoUML.

É importante notar que este Generalization Set *GS* precisa ser definido como apenas disjunto. Isso ocorre pelo fato do *Category X* estar generalizando obrigatoriamente entidades distintas.

4.2.8 Mixin

Um Mixin é um tipo semi-rígido que agrupa propriedades que são essenciais para algumas de suas instâncias e condicional as outras. Na Figura 51 abaixo apresentamos o procedimento para criação de Mixins no modelo. Note que para o caso dos *Mixins* temos uma abordagem

diferenciada. Para esses, é necessário que generalizem ao menos um sortal rígido e um anti-rígido de forma direta ou indireta, ou seja, também satisfaz a regra de generalização os casos em que um dado *Mixin X* generalize outros tipos de *Mixin Universal* (*RoleMixin*, *Category* ou *Mixin*) que por sua vez generalizarão sortais, que neste caso, deverão ter algum sortal rígido e anti-rígido como um de seus subtipos (também de forma direta ou indireta). Dessa forma, o procedimentos busca garantir que mesmo indiretamente alguns dos subtipos do *Mixin X* sejam ao menos um sortal rígido e um anti-rígido.

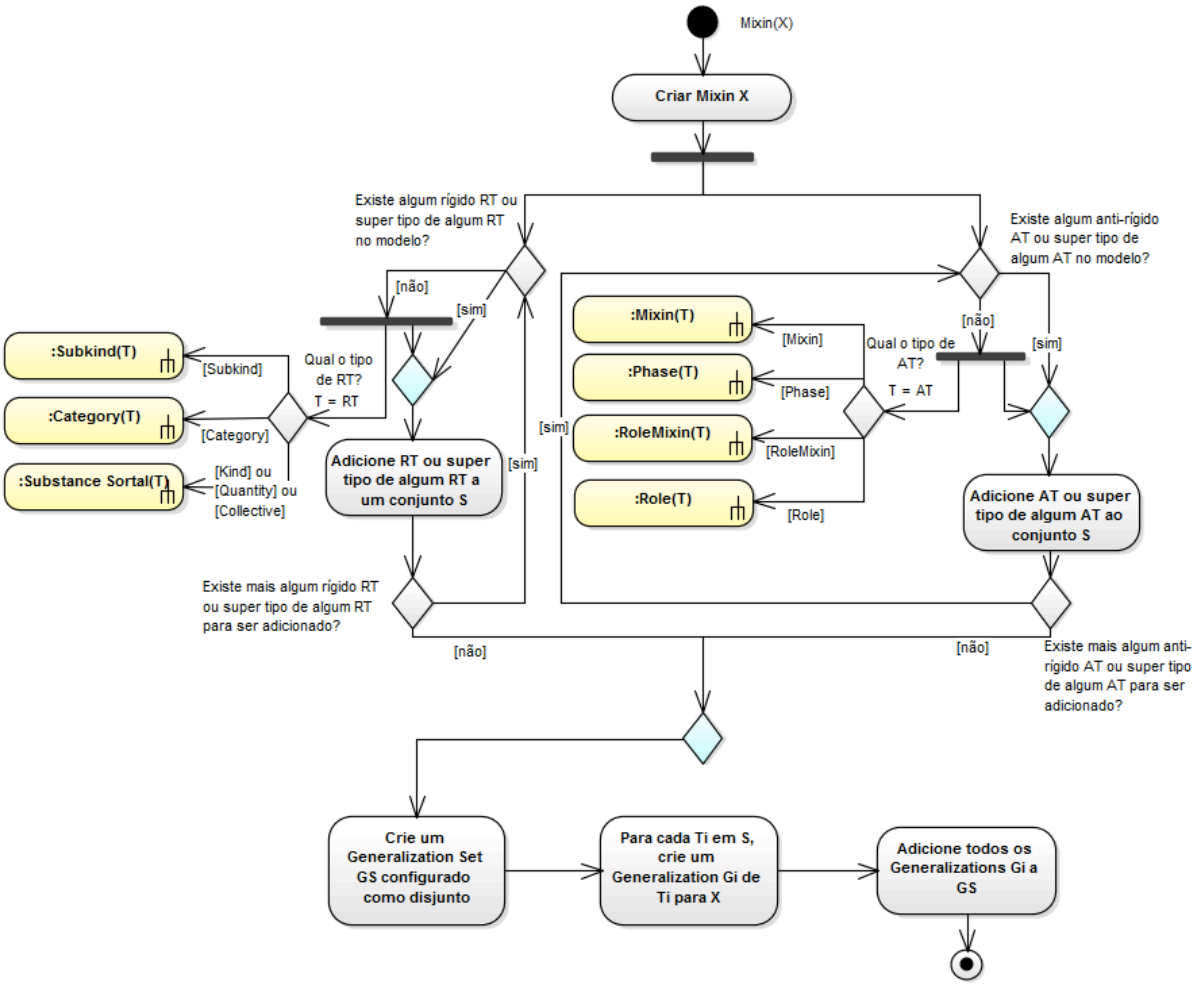

Figura 51 - Procedimento: Mixin

Como podemos ver é indicado ao usuário a seleção no modelo ou criação de um tipo rígido ou a seleção do supertipo de algum tipo rígido. Na sequência, repetimos o mesmo para seleção de um tipo anti-rígido ou de algum supertipo de algum tipo anti-rígido. Note que para ambos os casos é oferecido ao usuário a possibilidade de se adicionar outros tipos. Por fim,

um Generalization Set disjunto é criado e a ele é adicionado todos as generalizações dos subtipos para o *Mixin X*.

4.2.9 Mode

O tipo *Mode* é um *Moment* intrínseco o que significa que todas as suas instâncias são existencialmente dependentes de exatamente uma entidade. Por exemplo, o *Kind Person* pode ser caracterizado pelo *Mode Headache* (dor-de-cabeça). O que significa dizer que para cada instância de *Headache* temos uma instância específica e única de *Person* associada.

Por definição, *Modes* podem caracterizar qualquer *Endurant*. Dessa forma *Modes* podem ser usados, por exemplo, para designar a intensidade de outros *Modes*. Seguindo o exemplo anterior, poderíamos adicionar o *Mode Intensity* ao modelo e relacioná-lo com o *Mode Headache*, caracterizando a intensidade da dor-de-cabeça de alguma instância de *Person*. A Figura 52 abaixo descreve o processo de criação de um tipo *Mode*.

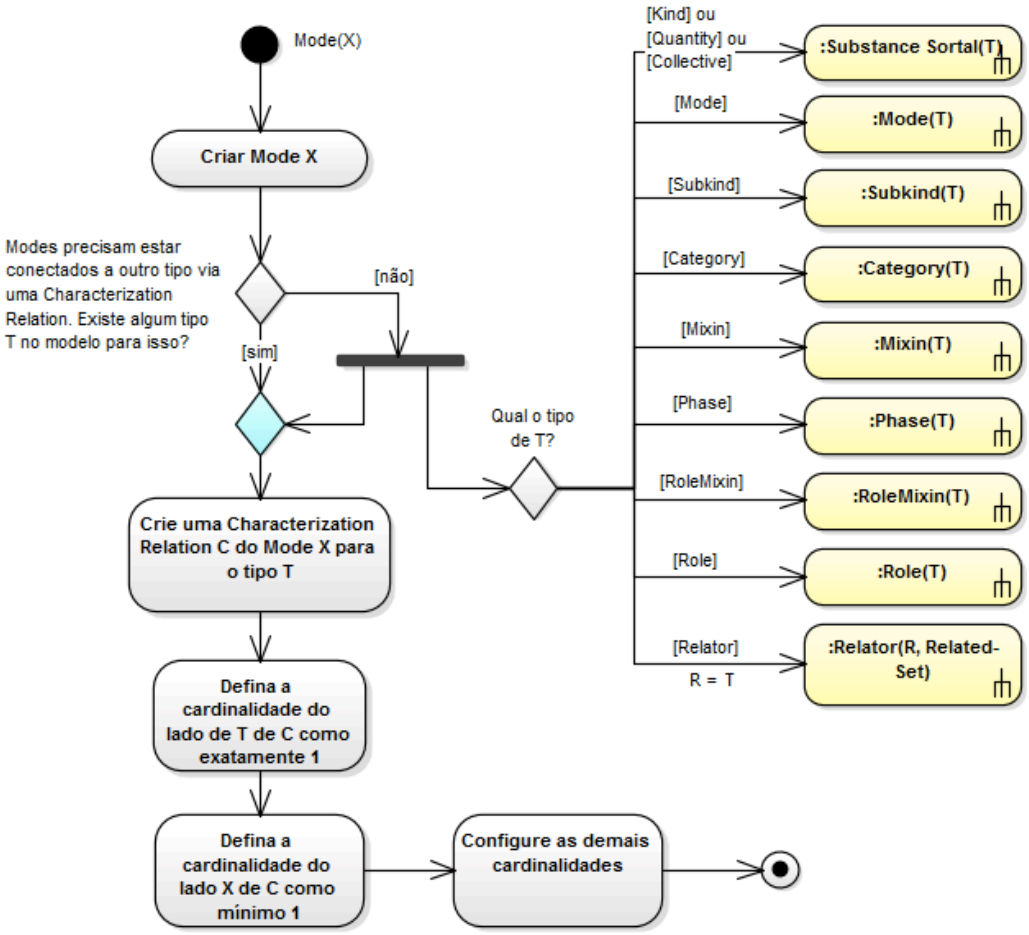

Figura 52 - Procedimento: Mode

Como visto na figura a cima, é forçado ao usuário a criação da relação de *Characterization*, bem como a definição das cardinalidades mínimas, como definido nas regras de fundamentação.

4.3 PROCEDIMENTOS RELACIONAIS

Diferente dos Procedimentos Básicos apresentados anteriormente, os Procedimentos Relacionais não possuem um conceito como entrada e não lidam diretamente com as regras de criação de novos tipos. Estes procedimentos são divididos entre os procedimentos de Partition, Generalization e Relation. Esses procedimentos lidam, respectivamente, com partições de tipo (a divisão de um tipo em dois ou mais subtipos em decorrência de uma propriedade comum para esta divisão), generalização (quando um tipo emerge do agrupamento de propriedades compartilhadas entre dois ou mais tipos) e relação (discute as possibilidades de relacionamentos entre os tipos e as cardinalidades que decorrem destes relacionamentos). Na sequência apresentamos cada um destes procedimentos.

4.3.1 Procedimento de Partição

A partição de um tipo é definida pela divisão desse tipo em subtipos que respeitam o mesmo princípio de partição. Na maioria dos casos, as partições acontecem quando existe um atributo no qual se deseja discutir sobre seus valores. Por exemplo, o atributo *gender* (gênero) do *Kind Person* pode ser necessário para alguns domínios discutir sobre seus possíveis valores (e.g. no domínio sobre casamento, pode-se definir uma relação de *casado-com* entre um homem e uma mulher). Para estes casos, pode ser necessário criar uma partição de *Person* em *Man* e *Woman*, porém para outros não, como nos domínios de aluguel de imóveis e empréstimo de livros, os quais pode não ser necessário discutir sobre os valores desse atributo. Ademais, uma vez que as partições buscam definir tipos para os possíveis valores de um atributo, temos que esses tipos devam ser obrigatoriamente do mesmo esterótipo.

Como demonstrando pelos padrões anteriormente descritos, existem vários arranjos possíveis para criação de partições, logo, todas as regras apresentadas anteriormente são levadas em consideração para construção desse procedimento para criação de partições em OntoUML. Entretanto, uma vez que a quantidade de possibilidades de combinações para todos os tipos de OntoUML se torna complexa (vide todos os padrões de partição apresentados em nosso catálogo), optamos por definir uma sequência de possibilidades apresentadas na Tabela 1 abaixo, onde *P* significa uma especialização possível e *F* onde não é possível.

Tabela 1 - Possibilidades de Partições

Estereótipos	Substance Sortals	Subkind	Role	Phase	Category	Mixin	Role Mixin	Relator	Mode
Substance Sortals	**F**	**P**	**P**	**P**	**F**	**F**	**F**	**F**	**F**
Subkind	**F**	**P**	**P**	**P**	**F**	**F**	**F**	**F**	**F**
Role	**F**	**F**	**P**	**P**	**F**	**F**	**F**	**F**	**F**
Phase	**F**	**F**	**P**	**P**	**F**	**F**	**F**	**F**	**F**
Category	**P**	**P**	**F**	**F**	**P**	**F**	**F**	**F**	**F**
Mixin	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
Role Mixin	**F**	**F**	**P**	**F**	**F**	**F**	**P**	**F**	**F**
Relator	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**P**	**F**
Mode	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**P**

A tabela resume as possibilidades de partições entre os tipos de OntoUML. Para cada linha, temos um possível estereótipo para ser o supertipo da partição, já as colunas representam o tipo dos possíveis subtipos deste supertipo. Para exemplificar, temos na primeira linha um *Substance Sortal* como supertipo e como possíveis subtipos, os tipos *Subkind*, *Role* e *Phase*. Como já explicado anteriormente, por se tratar de partições, todos os subtipos da partição devem ser classificados por apenas um estereótipo (motivo no qual não é possível a criação de partições com o tipo *Mixin*). O procedimento em sequência faz uso dessa tabela para assegurar a criação correta de uma partição.

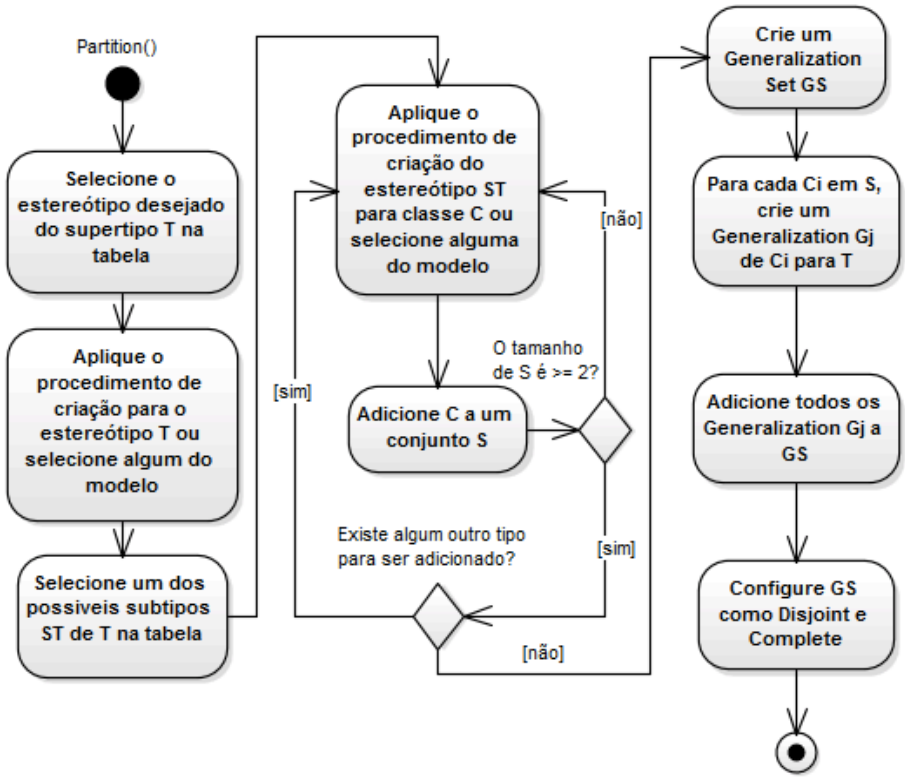

Figura 53 - Procedimento: Partition

O processo de criação de partições inicia com a seleção do supertipo e em seguida a definição de cada um de seus subtipos. Após isso, cria-se um Generalization Set disjunto e completo com todas as generalizações entre os subtipos e o supertipo. Por fim, verifica-se a criação de *Roles*, *RoleMixins* ou *Relators* no modelo, a fim de criar as relações de dependências desses tipos.

4.3.2 Procedimento de Generalização

Diferentemente do procedimento para criação de partições, o procedimento de criação de generalizações prevê que seu resultado será um fragmento de modelo onde o supertipo agrupa propriedades compartilhadas por seus subtipos. Um importante aspecto que distingue ambos os procedimentos é que para as generalizações, não é necessário que o estereótipo dos subtipos sejam todos do mesmo tipo. Uma vez que a generalização pressupõe a criação de supertipos a partir dos subtipos, não existe a possibilidade deste supertipo ser um *Sortal*, dado que este estaria provendo identidade para os subtipos. Para simplificar as possibilidades de generalização, apresentamos a Tabela 2, onde *P* significa uma especialização possível e *F* uma especialização não possível.

Tabela 2 - Possibilidades de Generalização

Estereótipo	Substance Sortals	Subkind	Role	Phase	Category	Mixin	Role Mixin
Category	**P**	**P**	**F**	**F**	**P**	**F**	**F**
Mixin	**P**	**P**	**P**	**P**	**P**	**P**	**P**
Role Mixin	**F**	**F**	**P**	**F**	**F**	**F**	**P**

Da tabela, temos que as linhas significam o tipo emergente (supertipo) e a coluna os possíveis subtipos. Na sequência apresentamos o procedimento para criação de Generalizations em OntoUML.

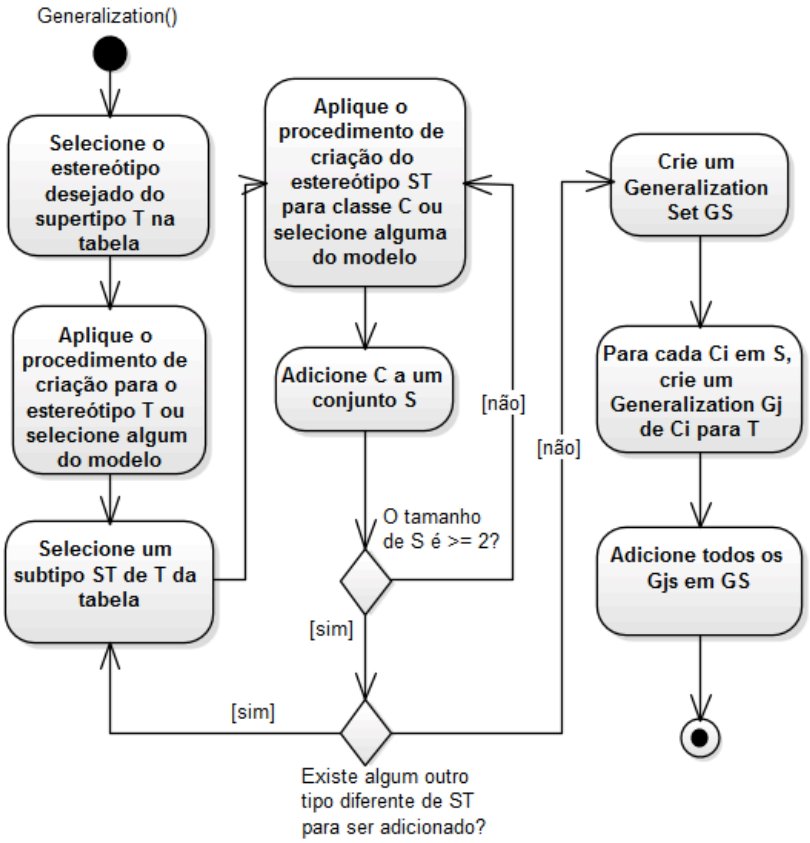

Figura 54 - Procedimento: Generalization

Além de permitir a seleção de mais de um estereótipo para o conjunto de subtipos da generalização, o procedimento a cima também mostra não ser necessário configurar como disjunto e completo a Generalization Set criada. Por fim, verifica-se a criação de *Roles*, ou *RoleMixins* no modelo, a fim de criar as relações de dependências desses tipos.

4.3.3 Procedimento de Relação

Em OntoUML temos uma grande quantidade de estereótipos para as relações, onde cada uma possui um propósito específico. Devido a isto, as relações são divididas entre *Dependency Relations* (Relações de Dependência) e *Meronymic Relations* (Relações de Todo-Parte). Enquanto no primeiro caso encontramos as relações *Material*, *Mediation*, *Characterization*, *Formal* e *Derivation*, nas relações de Todo-Parte temos as relações de *ComponentOf*, *SubQuantityOf*, *SubCollectionOf* e *MemberOf*. Devido suas particularidades, algumas relações só podem ocorrer entre tipos específicos da linguagem. Para simplificar isso, apresentamos a Tabela 3 abaixo onde é demonstrado os possíveis membros das relações de dependência.

Tabela 3 – Possibilidades das Relações de Dependência

Tipo/Relação	Formal	Material	Mediation	Characterization	Derivation*
Substance Sortal	**P**	**P**	**P**	**P**	**F**
Subkind	**P**	**P**	**P**	**P**	**F**
Role	**P**	**P**	**P**	**P**	**F**
Phase	**P**	**P**	**P**	**P**	**F**
Category	**P**	**P**	**P**	**P**	**F**
Mixin	**P**	**P**	**P**	**P**	**F**
Role Mixin	**P**	**P**	**P**	**P**	**F**
Relator	**P**	**F**	**M**	**P**	**M**
Mode	**P**	**F**	**F**	**M**	**F**

Na tabela a cima, temos as linhas representando os possíveis membros e as relações apresentadas nas colunas. Ademais, temos que P significa um membro possível da relação, F um membro proibido de fazer parte da relação e M um membro mandatório para relação. Os membros mandatórios são os tipos obrigatórios a existir na relação. Como podemos ver, a única relação com o comportamento especial é a relação de Derivation (asterisco). Isto ocorre por esta relação ter como seu membro mandatório um tipo Relator e como membro de destino uma relação Material. Por isso esta relação será tratada a parte do procedimento geral para criação das Relações de Dependência apresentado na Figura 55 abaixo.

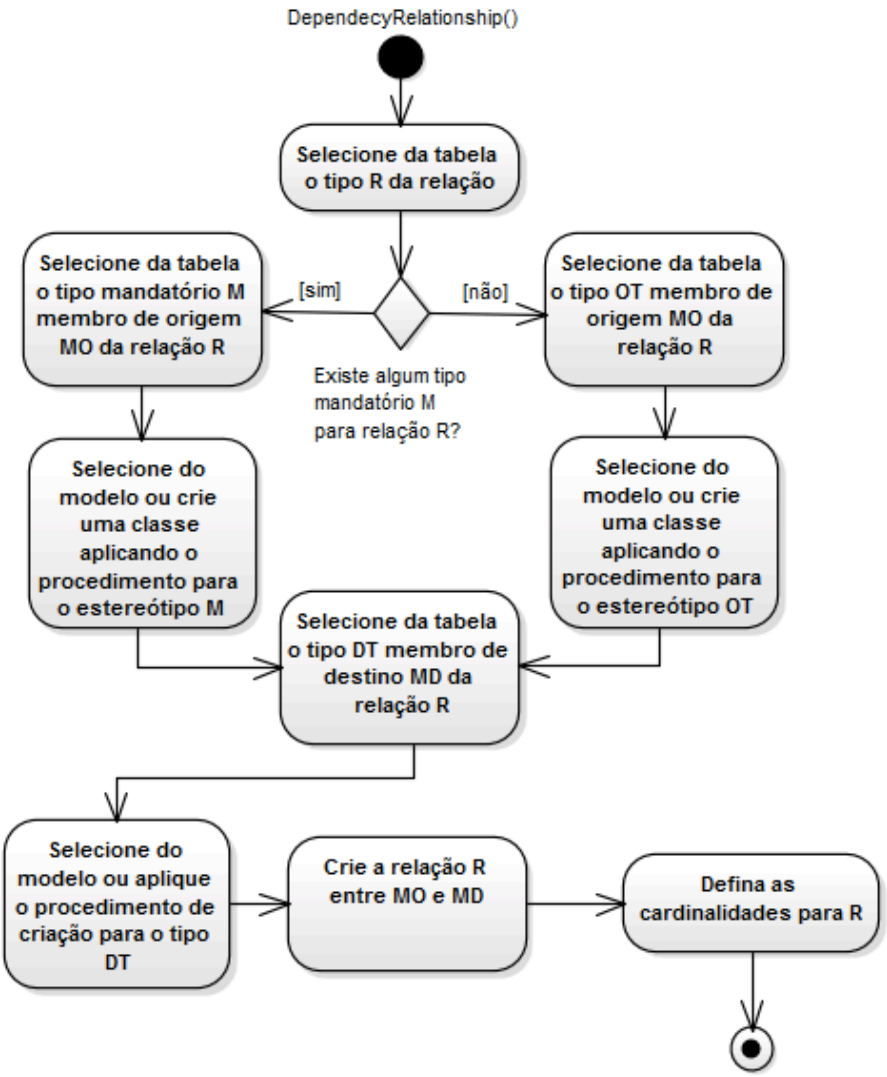

Figura 55 - Procedimento: Relações de Dependência

Primeiramente, identifica-se o tipo da relação a ser criada e depois a existência de algum tipo mandatório para esta relação. Para as relações com algum tipo mandatório, é necessário utilizá-los como membro origem da relação. Não havendo tipo mandatório, deve-se selecionar ou criar um novo elemento (utilizando o seu respectivo procedimento). Depois disso, para todos os casos, é indicado a seleção ou criação de um novo tipo destino e por fim, a definição da cardinalidade entre eles e a criação das relações de dependência para os casos de *Roles*, *RoleMixins* ou *Relators*. Nesse ponto, como já discutido anteriormente, a cardinalidades mínimas para estas relações é indicado ser maior que 1.

Como dito acima, a relação *Derivation* possuí um comportamento diferenciado das demais, uma vez que seu membro de origem é obrigatoriamente um *Relator* e o de destino uma relação *Material*. O procedimento descrito a seguir demonstra o processo de criação de uma relação *Derivation*.

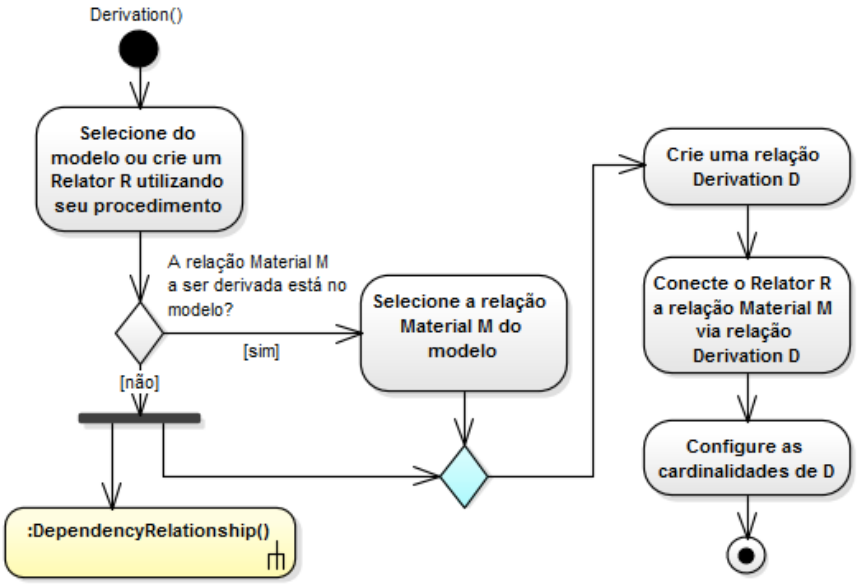

Figura 56 - Procedimento: Derivation

Nesse procedimento, primeiramente identificamos o *Relator* membro da relação e a relação *Material*. Caso ainda não exista a relação, deve-se cria-la utilizando o procedimento para criação de relações de dependência (DependencyRelationship). Por fim, cria-se a relação *Derivation* entre os dois e configure suas cardinalidades.

Como mencionamos anteriormente, o outro tipo de relações de UFO são as relações de Todo-Parte. Para distinguir entre os diferentes tipos de relacionamento entre as partes e seu todo, UFO defende a existência de quatro relações, i.e. *ComponentOf*, *SubQuantityOf*, *SubCollectionOf* e *MemberOf*. Assim como as relações de dependência, cada uma dessas possuí um propósito específico, dessa forma, não faz parte da intenção dos procedimentos apresentados aqui identificar quando se usar cada uma delas, mas sim como aplica-las. A Tabela 4 abaixo resume os possíveis arranjos de membros dessas relações.

Tabela 4 - Possibilidades das Relações Todo-Parte

Classe/Relação	Component Of	Sub Quantity Of	Sub Collection Of	Member Of
Kind	**P**	**F**	**F**	**P**
Quantity	**F**	**P**	**F**	**F**
Collective	**F**	**F**	**P**	**P**
Subkind	**R**	**R**	**R**	**R**
Role	**R**	**R**	**R**	**R**
Phase	**R**	**R**	**R**	**R**
Category	**R**	**R**	**R**	**R**
Mixin	**R**	**R**	**R**	**R**
Role Mixin	**R**	**R**	**R**	**R**

Assim como nas tabelas anteriores, o valor *P* significa a possibilidade de uso do estereótipo na relação e *F* que aquele estereótipo é proibido. Entretanto, para as relações todo-parte temos alguns casos de restrição para uso dos estereótipos, o que é representado pelo valor *R* na tabela. Estas restrições ocorrem porque algumas destas relações somente se relacionam com alguns *Substance Sortals* que estejam obedecendo determinadas regras (como demonstrado nas discussões para Figura 39).

Resumindo as regras de uso para as relações de Parte-Todo, temos que a relação de *ComponentOf* somente se relaciona entre complexos funcionais (GUIZZARDI, 2005). Nesse sentido, uma instância *X* é um complexo funcional se obedece alguma das seguintes regras:

- Se *X* é um Kind ou algum subtipo de algum Kind;
- Se *X* é algum Mixin Universal (i.e. Mixin, RoleMixin ou Category) cujo os subtipos são apenas Kinds ou subtipos de Kinds;

Estas regras se repetem para as relações de *SubQuantityOf* e *SubCollectionOf*, alterando-se o tipo da instância *X* de *Kind* para *Quantity* e *Collective* respectivamente. Logo, para a relação de *SubQuantityOf* temos:

- Se *X* é um Quantity ou algum subtipo de algum Quantity;
- Se *X* é algum Mixin Universal (i.e. Mixin, RoleMixin ou Category) cujo os subtipos são apenas Quantities ou subtipos de Quantities;

E temos para relação de *SubCollectionOf* as regras:

- Se *X* é um Collective ou algum subtipo de algum Collective;
- Se *X* é algum Mixin Universal (i.e. Mixin, RoleMixin ou Category) cujo os subtipos são apenas Collectives ou subtipos de Collectives;

Por fim, para a relação de *MemberOf* temos um caso mais restrito de possibilidades. Para essa relação temos que o lado do **todo** aplica as regras:

- Se *X* é o todo da relação, então *X* é um Collective ou algum subtipo de algum Collective;
- Se *X* é o todo da relação, então *X* é algum Mixin Universal (i.e. Mixin, RoleMixin ou Category) cujo os subtipos são apenas Collectives ou subtipos de Collectives;

Para o lado da **parte** de uma relação de *MemberOf* temos duas opções como descrito abaixo:

- Se *X* é uma das partes da relação, então *X* é um Kind ou algum subtipo de algum Kind;
- Se *X* é uma das partes da relação, então *X* é algum Mixin Universal (i.e. Mixin, RoleMixin ou Category) cujo os subtipos são apenas Kinds ou subtipos de Kinds;

Ou temos que:

- Se *X* é uma das partes da relação, então *X* é um Collective ou algum subtipo de algum Collective;
- Se *X* é uma das partes da relação, então *X* é algum Mixin Universal (i.e. Mixin, RoleMixin ou Category) cujo os subtipos são apenas Collectives ou subtipos de Collectives;

Como pode ser percebido pelas variações das regras para cada tipo de relação, temos que a aplicação de uma relação de todo-parte é uma tarefa complexa. Desta forma, o procedimento apresentado na Figura 57 em sequência busca simplificar esse processo.

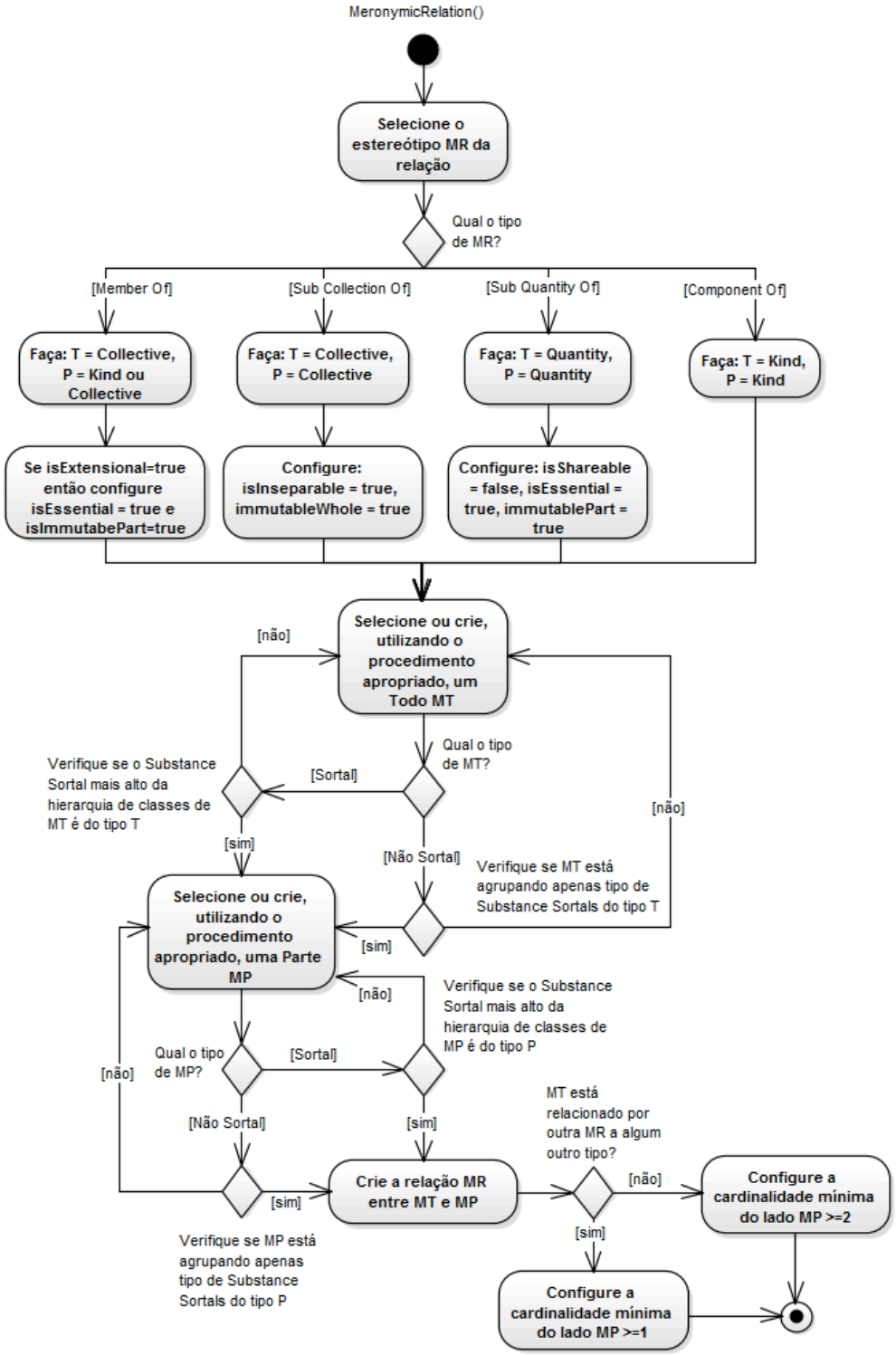

Figura 57 - Procedimento: Relações de Todo-Parte

A figura a cima deixa claro a complexidade de modelagem das relações de Todo-Parte. Nesse procedimentos, primeiro se identifica o estereótipo da relação a ser modelada, em sequência definimos as variáveis *T* e *P* as quais representam, respectivamente, as possibilidades de estereótipos para os tipos do Todo e da Parte da relação (como discutido acima para definição do tipo para instância *X*). Depois disso, definimos as meta-propriedades do tipo de relação

escolhido, conforme (GUIZZARDI, 2005), (GUIZZARDI, 2010) e (GUIZZARDI, 2011). Na sequência, cria-se os tipos para o Todo e para Parte verificando as regras demonstradas anteriormente para criação. Em seguida, temos a definição da cardinalidade da relação.

Como discutido em (GUIZZARDI, 2005) o tipo Todo de uma relação de Todo-Parte precisa estar ligado a ao menos duas instâncias de Parte, não necessariamente do mesmo tipo Parte. Em outras palavras, caso o Todo tenha apenas uma relação *R* de Todo-Parte com alguma Parte, a cardinalidade mínima desta relação terá de ser maior ou igual a 2. Caso o Todo tenha mais de uma relação do tipo *R*, é possível que cada uma tenha cardinalidade mínima maior ou igual a 1, ou seja, a soma das cardinalidades mínimas das relações do tipo *R* deverá ser maior que 2.

CAPÍTULO 5. AVALIAÇÃO DA ABORDAGEM

Como já mencionado anteriormente, o editor de ontologias OntoUML (OLED) é uma ferramenta com inúmeras funcionalidades para auxiliar o processo de modelagem de ontologias. Entretanto, não existia até o momento nenhuma funcionalidade na ferramenta para auxiliar a construção de ontologias com o uso de padrões. A iniciativa apresentada em (GUIZZARDI; GRAÇAS; GUIZZARDI, 2011) propõe uma ferramenta para construção de ontologias utilizando uma quantidade resumida de padrões em uma aplicação nova. No entanto, a ferramenta proposta naquela ocasião não possuía os outros recursos que OLED oferece. Diante disso, apresentamos neste capítulo, como forma de avaliação da nossa abordagem, a implementação dos padrões apresentados neste trabalho na ferramenta OLED, bem como os recursos de usabilidade e de auxílio a modelagem relacionados com esta implementação.

Além de avaliar a abordagem mediante sua implementação na ferramenta, esse capítulo também traz a execução de um exemplo de construção de um modelo OntoUML que utiliza os padrões de nosso catálogo, onde salientamos a forma de usá-los e avaliamos a adequação de seu uso a um caso real.

Por fim, apresentamos um algoritmo para identificação automática de incompletudes em modelos OntoUML e a solução dessas incompletudes utilizando os padrões de nosso catálogo. Esse algoritmo também serve como forma de avaliação de nossa abordagem, ao demonstrar que, como argumentado anteriormente, a abordagem desse trabalho pretende auxiliar na criação de modelos corretos segundo as regras de OntoUML.

5.1 APOIO AO USO DE PADRÕES ONTOUML EM OLED

Os padrões apresentados neste trabalho estão implementados de forma flexível para simplificar o uso por usuários novatos. Na figura abaixo temos a lista de padrões implementados na ferramenta.

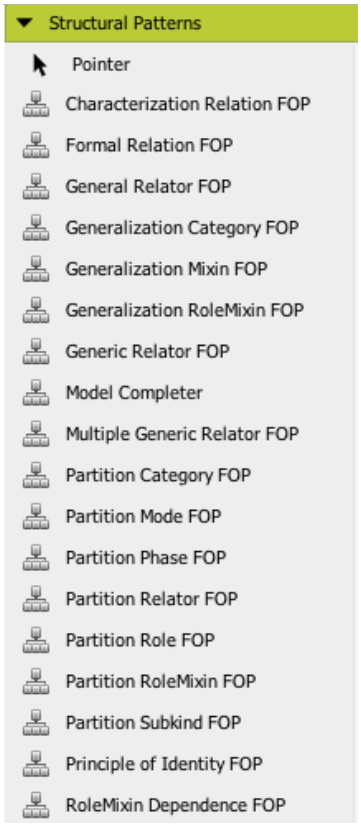

Figura 58 - Biblioteca de Padrões

Uma vez selecionado algum padrão da figura a cima, a janela abaixo é aberta possibilitando a customização do padrão escolhido para o domínio pretendido. Nessa funcionalidade, é possível customizar vários parâmetros do padrão, bem como reusar informações já existentes no modelo. A figura abaixo mostra com mais detalhes essa janela de configuração.

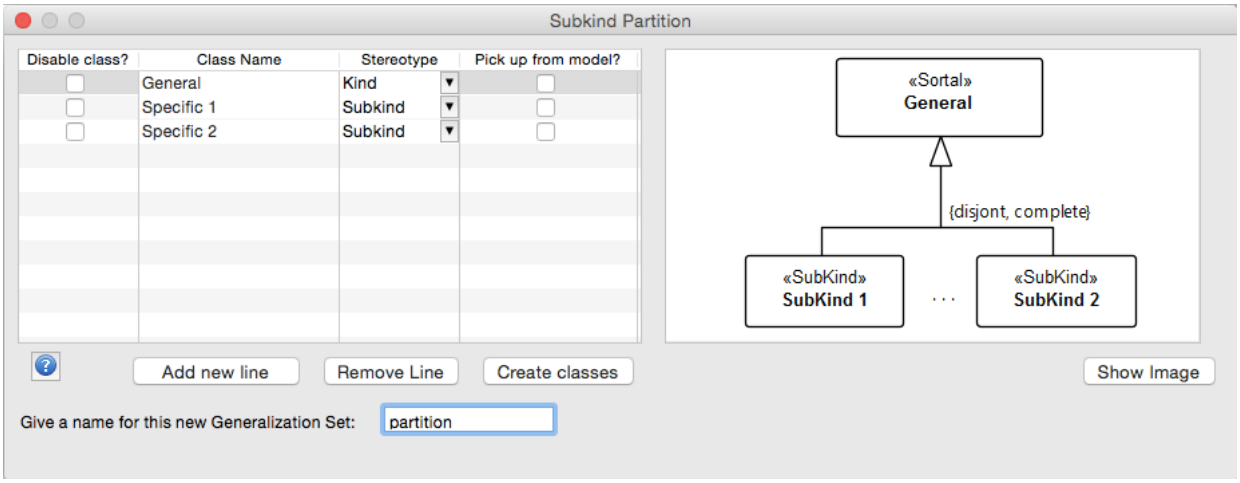

Figura 59 - Janela de Configuração de Padrão

Como pode ser visto, é possível customizar vários parâmetros do padrão por essa janela. Na primeira coluna o usuário pode optar por não utilizar uma classe do padrão, o que fará com que apenas uma parte do padrão seja aplicado ao modelo.

Na segunda coluna temos a opção de definir o nome para cada classe do padrão. Esse nome será usado na aplicação do padrão. Note que como demonstrado na figura, os nomes iniciais correspondem as respectivas classes no fragmento demonstrado a direita da janela.

Na terceira coluna temos uma lista de possibilidades para cada classe. Como nossos padrões utilizam classes abstratas de UFO para serem mais genéricos, deve-se substituir os estereótipos em questão por tipos específicos correspondentes da linguagem. Essa coluna serve exatamente para isso, aqui são mostrados a lista de possibilidades para cada classe do padrão, impossibilitando a seleção inadequada de algum tipo não válido para o padrão. Em outras palavras, esse campo impossibilita a customização errada do padrão, exibindo apenas os estereótipos válidos para cada classe do padrão.

Por último temos a quarta coluna onde é possível escolher uma classe já usada do modelo. Isso reduz o esforço durante o processo de modelagem, simplificando o reuso de classes já utilizadas e tornando mais dinâmico a aplicação dos padrões. Vale ressaltar que ao selecionar uma classe do modelo, os campos apresentados nas colunas anteriores se tornam não editáveis, uma vez que não cabe a essa janela de configuração permitir edição de classes já utilizadas no modelo.

Para auxiliar a construção de partições e generalizações, criamos os botões de adicionar nova linha (*Add new Line*) e de remover linhas (*Remove Line*). Estes botões servem para adicionar novas classes a um padrão de partição e generalização, facilitando a construção de taxonomias nos modelos.

Outra funcionalidade implementada visando aprimorar a flexibilidade da aplicação dos padrões foi o reuso de *Generalization Sets*. Essa funcionalidade serve para editar partições e generalizações. Uma vez aplicado um padrão desse tipo, é possível selecioná-lo do modelo (identificado o nome do Generalization Set utilizado no momento da aplicação do padrão) para adicionar ou remover elementos do Generalization Set utilizando a janela de configuração de padrão (canto inferior da janela da Figura 59).

Por último, criamos um botão de ajuda para usuários novatos da linguagem. Esse botão aparece a esquerda inferior da janela de configuração de padrão e mostra informações sobre cada um dos possíveis estereótipos do padrão corrente, como mostrado na figura abaixo.

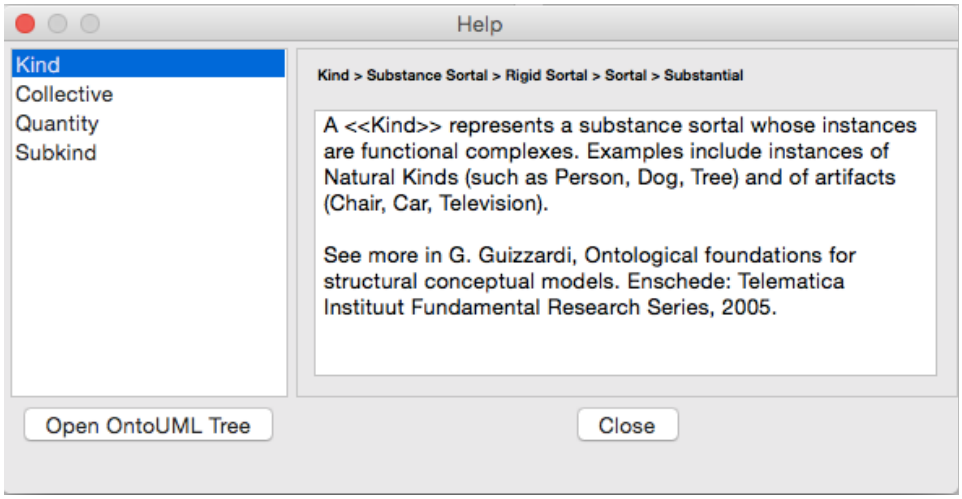

Figura 60 - Janela de Ajuda

Como pode ser visto na figura acima, apenas os tipos envolvidos no padrão corrente são mostrados com suas respectivas definições extraídas de (GUIZZARDI, 2005), bem como uma opção de visualizar a taxonomia de tipos de UFO (botão *Open OntoUML Tree*), i.e. Figura 1 apresentada no início desta dissertação.

5.2 EXEMPLO DE APLICAÇÃO: UNIVERSIDADE

Após descritos os padrões de OntoUML em nosso catálogo e apresentados os procedimentos para construção de modelos, apresentamos nesta seção um exemplo sob o domínio de Universidade (*University*) realizado na ferramenta OLED e utilizando os padrões apresentados em nosso catálogo e implementados na ferramenta. Escolhemos esse domínio por ser um domínio neutro com relação a aspectos culturais e de amplo conhecimentos da comunidade cientifica. A intenção é demonstrar em um caso prático a aplicação dos padrões de OntoUML. Abaixo apresentamos uma breve descrição do domínio.

Universidades possuem estudantes e professores. Nestas universidades, professores não podem ser estudantes, nem estudantes podem ser professores. O vínculo entre a universidade e seus estudantes é feito pela matrícula. Os professores possuem contrato de trabalho com a universidade. As aulas da universidade são lecionadas por um professor para vários alunos. Dos professores e alunos deseja-se saber quantos são homens e quais são mulheres, bem como a faixa etária (se jovem ou adulto).

Diante desta pequena descrição do domínio iniciamos nosso modelo. Primeiramente, vamos a primeira partição identificada no domínio, entre Pessoa (*Person*) e Estudante (*Student*) e Professor (*Professor*).

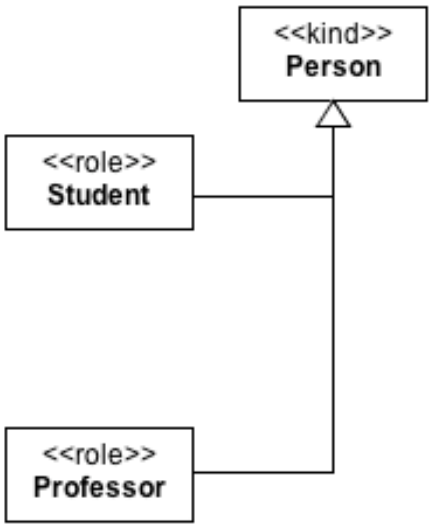

Figura 61 - Exemplo - Parte I

Aplicando-se o padrão Role Partition (Figura 23) temos o fragmento acima. Como especificado na descrição do domínio, Professores não podem ser Estudantes, nem vice-versa, logo, modelamos essas classes disjuntas entre si. Deste fragmento temos que tanto *Student* quanto *Professor* necessitam ter suas dependências relacionais resolvidas. Para isto, aplicamos o Generic Relator FOP (Figura 18) ao modelo.

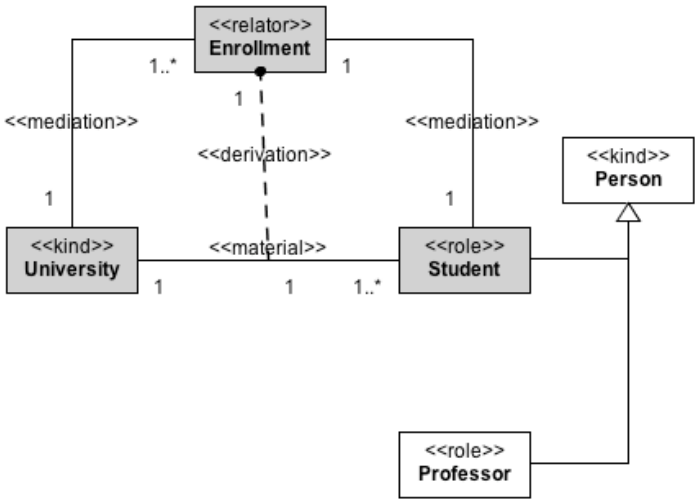

Figura 62 - Exemplo: Parte II

Nesse segundo fragmento destacamos as classes modificadas ou criadas pela aplicação do Generic Relator FOP (classes em cinza). Como descrito no texto de domínio do problema, o elemento que sustenta a relação entre *Student* e uma Universidade (*University*) é a Matrícula (*Enrollment*). Desta forma, criaremos este elemento como sendo o *Relator* que conecta ambas as entidades. Ademais, modelamos o conceito *University* como sendo do tipo *Kind*. O próximo passo é satisfazer o relacionamento da classe *Professor*, como mostrado na figura abaixo.

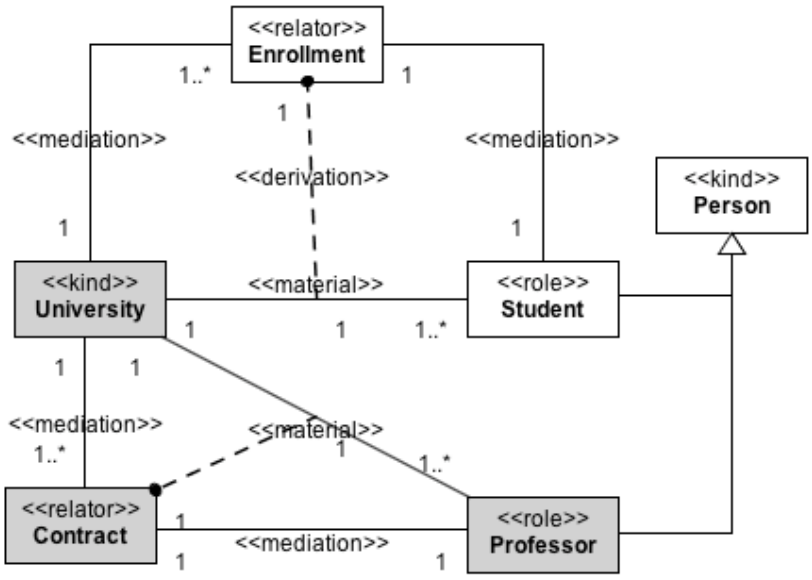

Figura 63 - Exemplo: Parte III

Nesse terceiro fragmento temos em destaque a aplicação do Generic Relator FOP. Dessa vez aplicamos esse padrão para definir a relação entre *Professor* e *University*. Como dito na

descrição do domínio o que sustenta a relação entre essas classes é um Contrato (*Contract*) entre elas. Dessa forma, aplicamos o padrão tendo o *Relator* como a classe *Contract*.

Ademais, como descrito no domínio do problema, as aulas da universidade são lecionadas de um professor para vários alunos. Uma vez que temos três membros relacionados, a aplicação do Multiple Generic Relator FOP (Figura 17) se faz mais adequada.

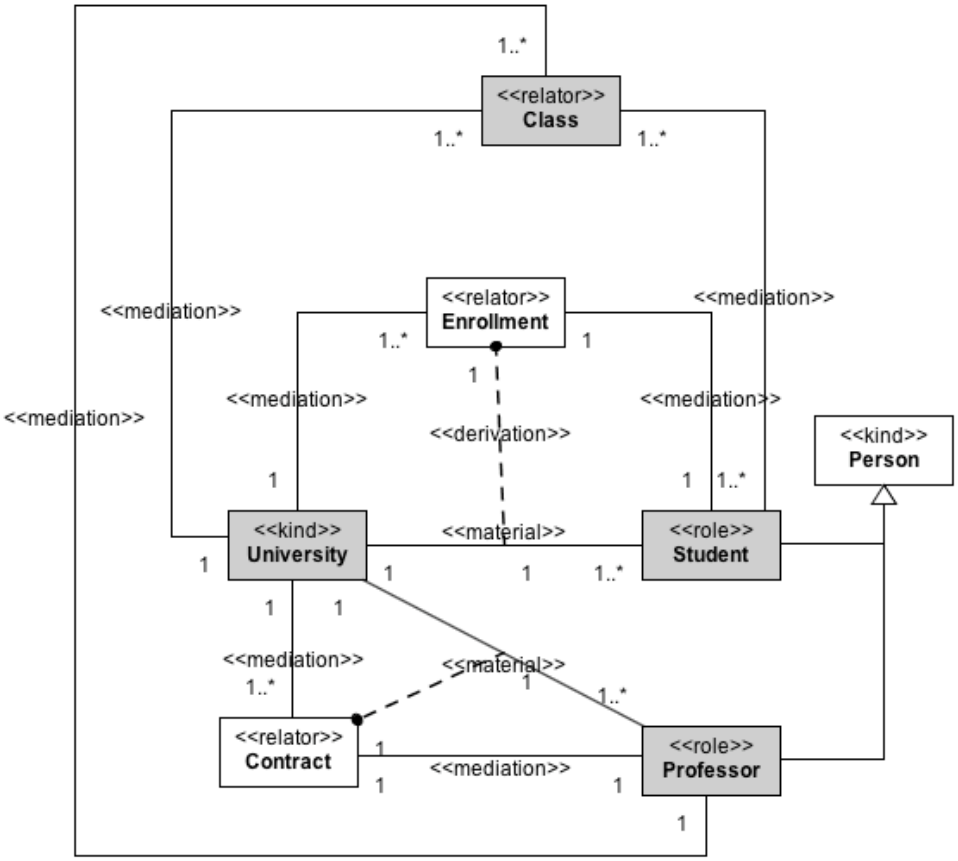

Figura 64 - Exemplo: Parte IV

Como pode ser visto na figura acima, temos relacionado as classes *Professor, University* e *Student* com o *Relator Class* (Aula), dessa forma, conseguimos saber de uma dada aula todos os alunos, o professor responsável e sua respectiva universidade.

Além disso, também temos que dos Professores e Alunos deseja-se saber quais são homens e quais são mulheres. Para isso, podemos atribuir uma nova partição a classe *Person* caracterizada pela distinção quanto aos gêneros Homem (*Man*) e Mulher (*Woman*). Para esse caso, aplicamos o Subkind Partition FOP (Figura 25) ao modelo, resultando na figura abaixo.

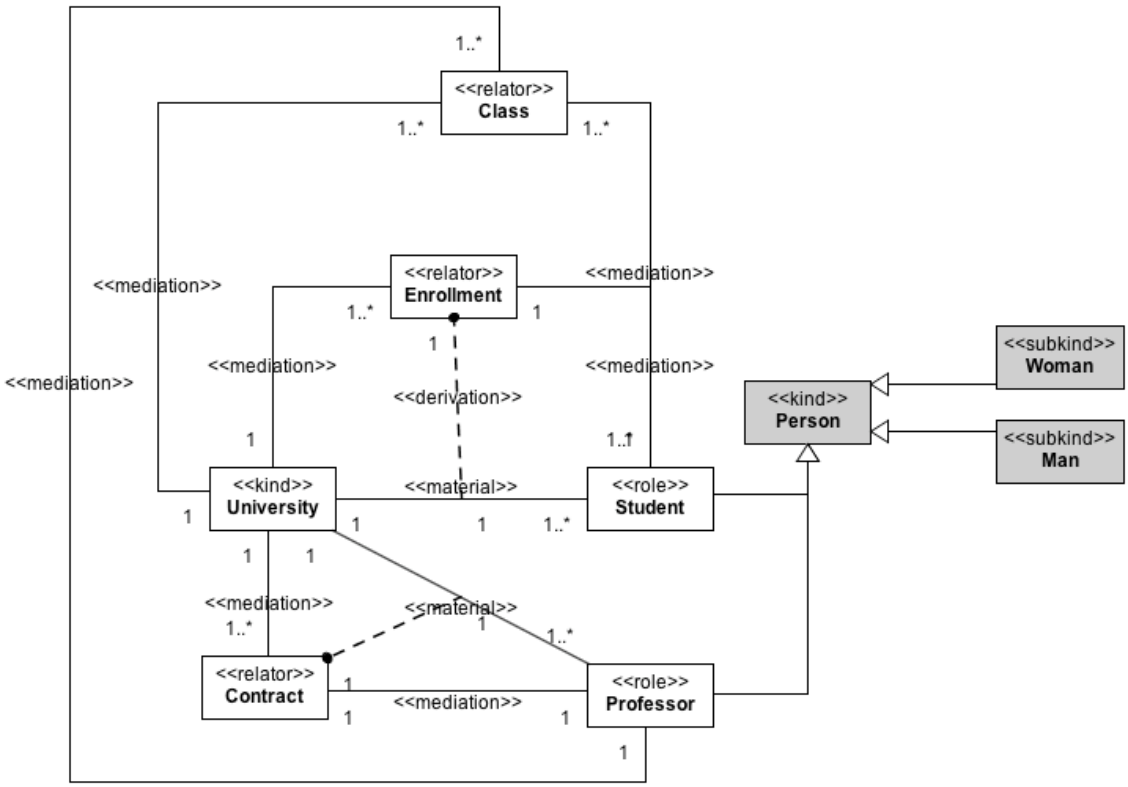

Figura 65 - Exemplo: Parte V

Nessa Parte V do modelo, temos duas partições para a classe *Person*, logo, para qualquer instância de *Person* no modelo, temos que ela será ou *Man* ou *Woman* **e** ou *Student* ou *Professor*. Além da partição por gênero, também temos a distinção quanto a faixa de idade (jovem ou adulto). Nesse caso, a figura abaixo mostra a aplicação do padrão Phase Partition FOP (Figura 24) ao nosso modelo.

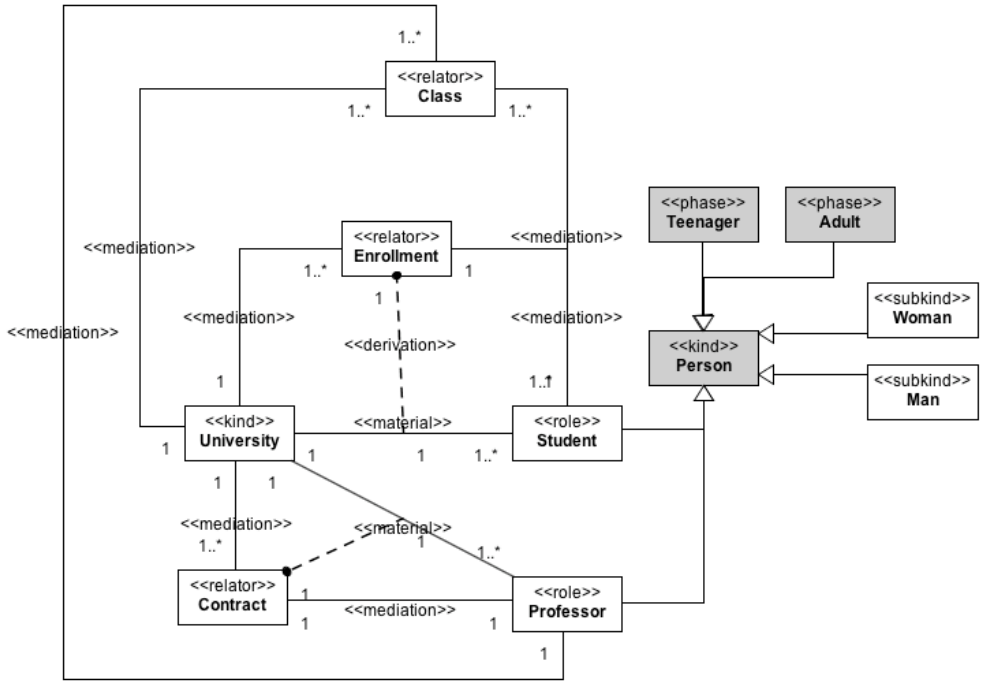

Figura 66 - Exemplo: Parte VI

Por fim, temos na Figura 66 o modelo completo proposto para o domínio modelado. Como pode ser visto, apenas com a recorrente aplicação de padrões conseguimos montar um modelo completo e sintaticamente válido, o que atesta que o uso dessa abordagem pode prover a criação de modelos OntoUML completos e válidos. Entretanto, acreditamos que futuros trabalhos possam ser realizados para certificar de que abordagem proposta é consistente (i.e. qualquer modelo criado pela abordagem é válido) e completa (i.e. qualquer modelo OntoUML válido pode ser construído pela abordagem).

5.3 ALGORITMO DE VERIFICAÇÃO DO ATENDIMENTO DE RESTRIÇÕES EM MODELOS ONTOUML

O Algoritmo de Verificação do Atendimento de Restrições em Modelos OntoUML tem por objetivo auxiliar na checagem e correção de modelos incompletos (i.e. que possui alguma regra de fundamentação ainda não resolvida). Nesse algoritmo, procuramos juntar o catálogo de padrões com a utilização dos procedimentos, provendo uma forma na qual tanto usuários novatos quanto avançados possam usufruir dos benefícios de ambas as abordagens.

Esse algoritmo está implementado na ferramenta OLED e utiliza os padrões apresentados aqui para solucionar as incompletudes encontradas nos modelos. Em seu funcionamento

temos que ao se identificar uma incompletude no modelo, uma lista dos padrões de nosso catálogo é indicado para, quando aplicado, solucionar esta incompletude. Vale destacar, que a aplicação de um novo padrão pode gerar o surgimento de uma nova incompletude no modelo, logo, implementamos na ferramenta OLED a possibilidade de se verificar o modelo a cada adição de um novo tipo ou aplicação de um novo padrão. Na Figura 67 em sequência vemos a visão geral do comportamento desta nova funcionalidade.

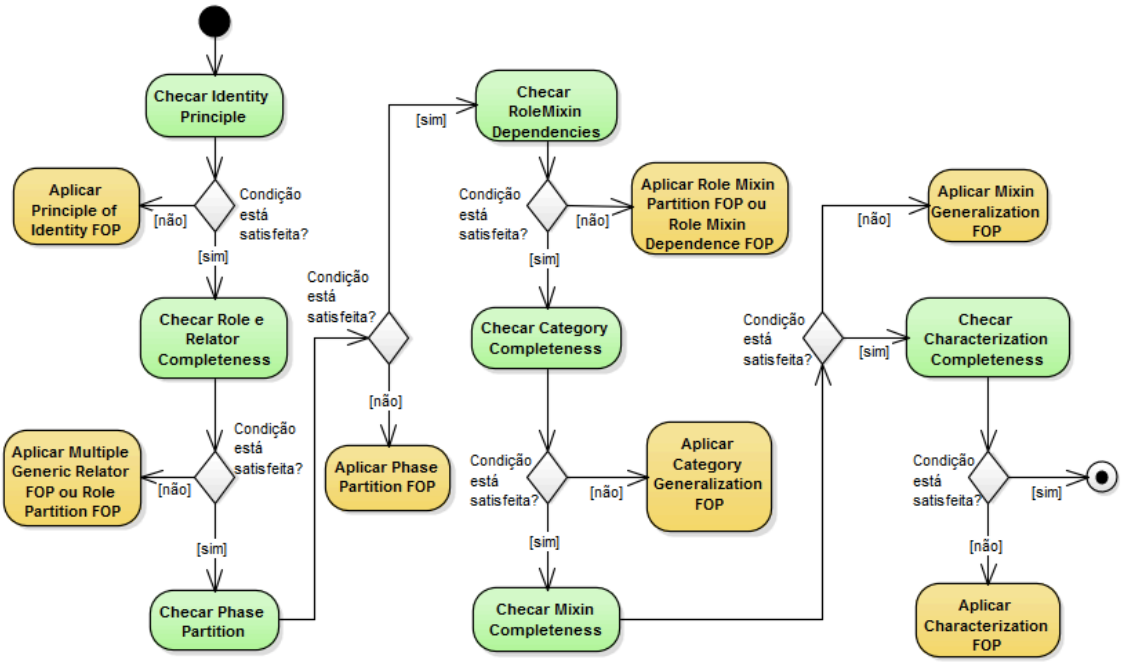

Figura 67 - Algoritmo de Verificação das Restrições em Modelos OntoUML

Essa visão geral nos permite verificar o fluxo de execução geral deste algoritmo e os tipos de checagem que ele executa (caixas verdes). Além disso, também podemos ver os padrões que podem ser aplicados em cada situação (caixas amarelas). Vale ressaltar, que a sequência de atividades é arbitraria e que as condições de checagem poderiam ser realizadas em paralelo. Ademais, esse algoritmo visa aplicar as checagens das mesmas regras de fundamentação já apresentadas anteriormente, as quais originaram tanto os padrões quanto os procedimentos.

Essa funcionalidade demonstra que a abordagem adota na execução desse algoritmo da suporte aos usuários novatos indicando quais padrões aplicar em cada evento de modelagem de forma automática. Para ilustrar este comportamento, vamos supor a inserção de uma classe *R* estereotipada como *Role* em um modelo OntoUML vazio. Automaticamente, a tela abaixo será mostrada ao usuário informado as incompletudes encontradas e os possíveis padrões a serem usados para solução.

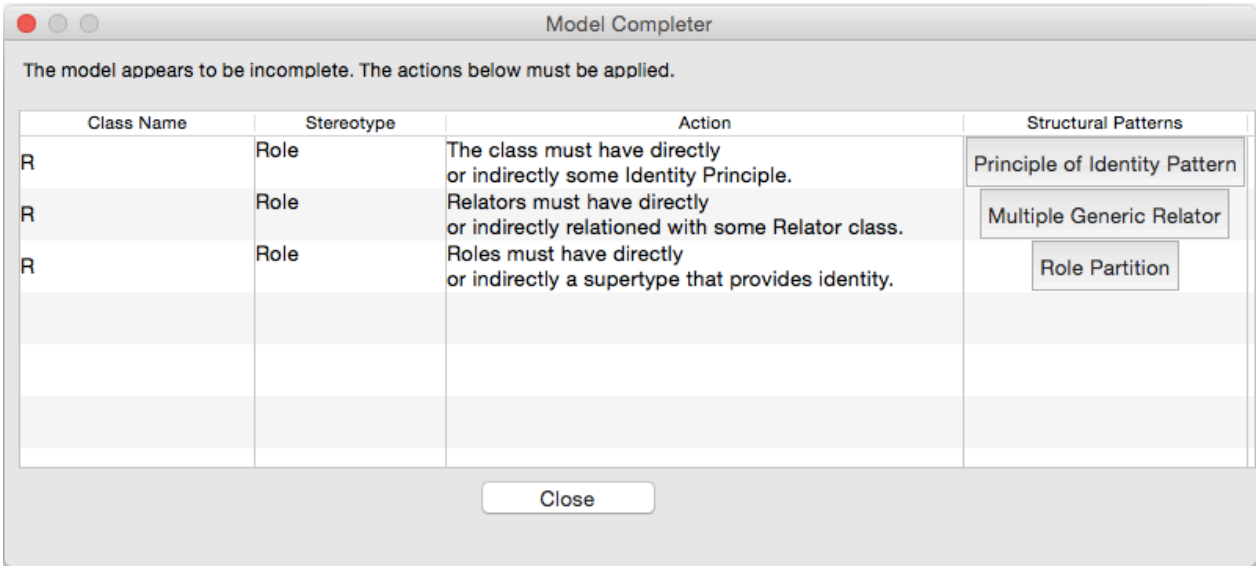

Figura 68 - Tela de Verificação do Modelo

A criação de um novo elemento no modelo resulta na verificação de completude de todas as regras do modelo. No exemplo da Figura 68 temos que a aplicação do algoritmo para o *Role R* identificou a necessidade de se criar uma classe que o forneça identidade, bem como criar um relacionamento entre o *Role R* e algum *Relator*.

Como pode ser visto, a aplicação deste algoritmo traz benefícios de usabilidade para o processo de modelagem. Por exemplo, a identificação e sugestão de solução para as incompletudes do modelo os quais servem de apoio para identificação das regras de fundamentação que faltam ser resolvidas. Além disso, adicionamos uma nova funcionalidade a tela de aplicação de padrões. Uma vez que o usuário seleciona qual padrão a ser inserido, identificamos na tela de aplicação para o padrão selecionado a classe que originou a ação. Para exemplificar, vamos supor que na tela da Figura 68 seja escolhido a aplicação do padrão Multiple Generic Relator (Figura 17), logo a Figura 69 abaixo demonstra como seria a tela de aplicação deste padrão.

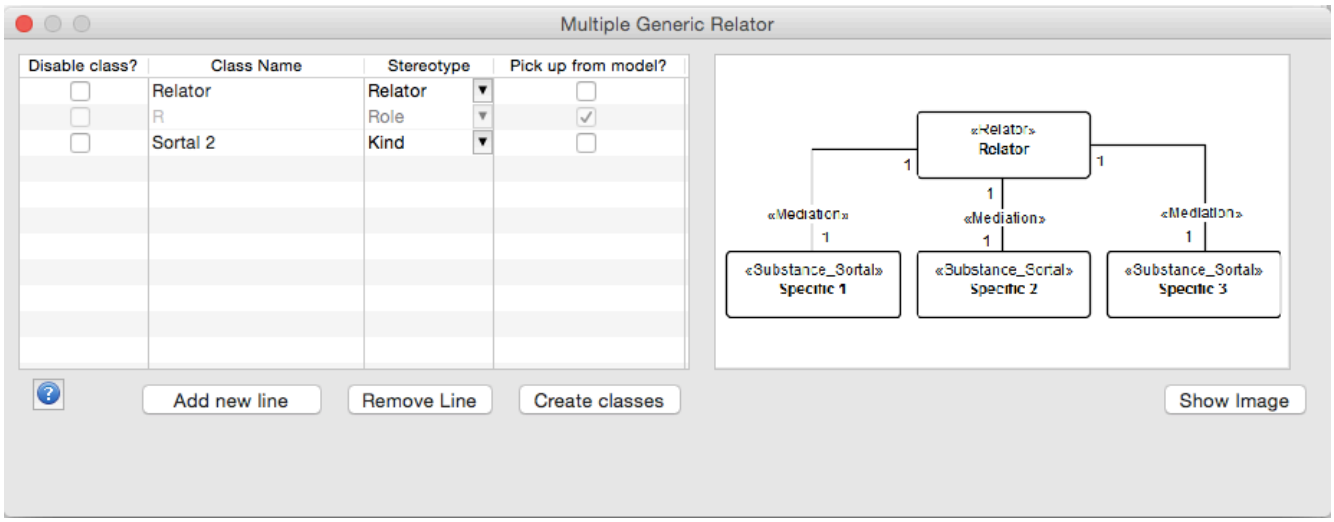

Figura 69 - Aplicando o Padrão Multiple Generic Relator

Na figura acima temos que o *Role R* encontra-se destacado na aplicação do padrão. Uma vez que esta foi a classe que originou a aplicação deste padrão, ela não pode ser modificado nessa tela, encontrando-se não editável para o usuário.

Por fim, temos que a aplicação de um novo padrão (e.g. Multiple Generic Relator) irá resultar em uma nova verificação do modelo pelo algoritmo, o que pode resultar em novas regras para completude do modelo, desta forma, realizando sucessíveis aplicações de padrões no modelo, visando mantê-lo sempre alinhado com as regras de fundamentação de UFO.

5.4 TRABALHOS RELACIONADOS

O uso das meta-propriedades de UFO para auxiliar na construção de modelos OntoUML já foi experimentado por outros trabalho. Assim como em nosso trabalho, em (GRAÇAS, 2010) é apresentado uma análise das regras de fundamentação de UFO visando a construção de procedimentos e padrões para auxiliar no processo de modelagem utilizando OntoUML. Entretanto, naquele trabalho apenas os tipos *Kind*, *Subkind*, *Phase* e *Role* foram estudados. Além disso, a análise realizada para o tipo *Role* leva em consideração a dependência relacional do tipo, não considera a representação explicita de *Relators*. Diante disso, o presente trabalho pode ser visto como uma extensão do trabalho realizado por aqueles autores, porém adicionando os demais tipos da linguagem e realizando uma analisa dos procedimentos de criação das relações.

Outro importante aspecto apresentado em nosso trabalho é a implementação de todos os padrões de nosso catálogo na ferramenta OLED. Em (GUIZZARDI; GRAÇAS; GUIZZARDI, 2011) é apresentado uma implementação em uma nova ferramenta, porém essa nova ferramenta não contava com todos os outros trabalhos presentes no OLED. Diante disso, decidimos por escolher implementar as novas funcionalidades diretamente no OLED ao invés usar a antiga ferramenta.

Por fim, temos que a adição dos novos padrões e procedimentos, bem como o algoritmo para verificação de modelos OntoUML formam um conjunto importante de melhorias nas ideias apresentadas previamente naquele trabalho, caracterizando o presente trabalho como uma extensão das ideias apresentadas anteriormente.

CAPÍTULO 6. CONSIDERAÇÕES FINAIS E PERSPECTIVAS FUTURAS

6.1 CONSIDERAÇÕES FINAIS

Esta dissertação tinha por objetivo estender a abordagem anterior para construção de modelos OntoUML baseada em padrões. Visando atingir este objetivo, primeiramente, propomos a criação de um Catálogo de Padrões de OntoUML para servir como guia para auxiliar aos modeladores. Para realizar essa tarefa, analisamos os padrões já identificados na literatura, como os apresentados em (GUIZZARDI, 2005) e (GUIZZARDI; GRAÇAS; GUIZZARDI, 2011). Ademais, realizamos uma analise das regras de fundamentação de UFO para identificar a existência de novos padrões. Por fim, apresentamos nosso catálogo de padrões de forma estruturada. Entretanto, para auxiliar no uso dos padrões apresentados, implementamos todo o nosso catálogo na ferramenta OLED, a qual já conta com uma vasta variedade de funcionalidades para auxiliar no processo de modelagem utilizando a linguagem OntoUML. Dessa forma, nosso trabalho passa a fazer parte de um arcabouço de trabalhos, que unidos, contribui para a realização de um suporte metodológico e computacional na construção de modelos OntoUML.

Após a identificação e formalização dos padrões em nosso catálogo, se fez necessário propor uma abordagem de modelagem para usuários novatos da linguagem. Dessa forma, propomos uma serie de procedimentos a serem usados com este objetivo. Esses procedimentos tendem, por fim, gerar o mesmo resultado do uso dos padrões. Além disso, os procedimentos prezam pela recorrência de aplicação. Em outras palavras, a aplicação de um procedimento gera a chamada a outros procedimentos de forma recorrente, possibilitando ao usuário perceber a cada chamada as regras que estão sendo aplicadas.

Uma vez formalizado nosso catálogo de padrões e os procedimentos para construção de modelos OntoUML, percebemos a necessidade de se criar uma abordagem mista dessas duas técnicas. Para isso, criamos um algoritmo para verificação de modelos OntoUML na ferramenta OLED. Esse algoritmo utiliza um procedimento de execução específico para verificar as incompletudes do modelo e propor a aplicação de padrões para solução desses

problemas. Dessa forma, para cada inserção de um novo tipo no modelo é mostrado ao usuário uma lista de padrões para serem usados visando solucionar os problemas identificados. Por fim, apresentamos um exemplo no qual aplicamos os padrões de nosso catálogo para solucionar um problema do domínio universitário. Ademais, confrontamos o presente trabalho com outros trabalhos relacionados na literatura.

Dessa forma concluímos nosso trabalho apresentando as principais contribuições desta dissertação:

i. Definição de um Catálogo de Padrões para OntoUML;
ii. Descrição dos procedimentos para aplicação desses padrões;
iii. Implementação do Catálogo de Padrões na ferramenta OLED;
iv. Implementação do algoritmo de verificação de modelos em OntoUML.

Vale ressaltar que neste trabalho realizamos uma análise quanto a aplicabilidade de *i* e *ii*, deixando possibilidade para trabalhos futuros analisarem, de maneira formal, a completude e consistência dos modelos gerados utilizando os padrões e procedimentos. Para *iii* e *iv* acreditamos que trabalhos futuros (como os discutidos em sequência) possam servir para analisar os benefícios do uso dessas funcionalidades na ferramenta OLED e no processo de modelagem.

6.2 PERSPECTIVAS FUTURAS

Considerando o estágio atual aqui apresentado, algumas das perspectivas de trabalhos futuros são destacadas a seguir.

Mesmo defendendo a tese de que o uso dos padrões e procedimentos podem auxiliar na melhoria do processo de modelagem, um estudo empírico contrastando a modelagem clássica de OntoUML (sem o auxílio de padrões e procedimentos) e a abordagem proposta aqui deve ser executado. Esse estudo empírico deve avaliar as questões empíricas pressupostas no desenvolvimento desta dissertação, são eles: obter um feedback dos usuários quanto as melhorias propostas tanto ferramental quanto do uso das abordagens (com padrões e/ou com procedimentos), analisar se um grupo de usuários (misto de novatos e avançados) utilizando a abordagem proposta aqui conseguem de fato reduzir o tempo de modelagem, além de analisar

a acurácia dos modelos quando comparados com um outro grupo de usuários (misto de novatos e avançados) utilizando a abordagem clássica de modelagem (OntoUML sem os padrões ontológicos e sem os procedimentos).

Como apresentado em (RUY et al., 2011), acreditamos que o uso de padrões de domínio (DROPs) possa ser melhor explorados, verificando como o processo de engenharia de ontologias possa ser aprimorando com a aplicação combinada de FOPs e DROPs. Essa abordagem se sustenta com o surgimento de novas Core Ontologies, as quais proveem uma variedade de DROPs a serem usados em domínios específicos e que poderão ser usados de forma combinada com os FOPs apresentados em nosso catálogo.

Por fim, uma vez que este trabalho seja uma extensão do trabalho de (GRAÇAS, 2010), trabalhos futuros poderiam identificar como as demais contribuições daquele trabalho se relacionaria com as apresentadas aqui. Destacamos a técnica de recomendação de estereótipo, a qual realiza uma série de questionamentos ao usuário sobre o conceito introduzido no modelo a fim de procurar identificar qual seria o estereótipo mais indicado para ele. Nesse aspecto, um futuro trabalho poderia adicionar essa técnica de recomendação a ferramenta OLED e conectá-la com os padrões apresentado em nosso catálogo, dessa forma, após a identificação do estereótipo, seria apresentado ao usuário as opções de padrões a serem aplicados. Além disso, em Graças também introduz um Modelo de Raciocínio sobre Padrões de Projetos, o qual poderia servir para coletar as informações de uso dos padrões apresentados em nosso catálogo (o que também deveria ser implementado na ferramenta OLED) e serviria (como defendido em (GRAÇAS, 2010)) para divulgação, para outros modeladores, das decisões de modelagem tomadas durante o processo de construção do modelo.

REFERÊNCIAS

BARCELOS, P. P. F. **Análise Arquitetural, Ontológica e Proposta de Modelo de Referência para a Recomendação ITU-T G. 805**. 2011.

BARCELOS, P. P. F., DOS SANTOS, V. A., SILVA, F. B., MONTEIRO, M. E., & GARCIA, A. S. **An Automated Transformation from OntoUML to OWL and SWRL**. In ONTOBRAS (pp. 130-141), 2013.

BRAGA, B., ALMEIDA, J.P.A., GUIZZARDI, G.; BENEVIDES, A.B. **Transforming OntoUML into Alloy: Towards Conceptual Model Validation using a Lightweight Formal Method**. Innovations in System and Software Engineering (ISSE), Springer-Verlag, 2010.

BODART, F.; PATEL A.; SIM M.; WEBER R. **Should optional properties be used in conceptual modelling? A theory and three empirical tests. Information Systems Research**. Information Systems Research, 2001, pp.384-405.

BRINGUENTE, A.C. ; FALBO, R. A. ; GUIZZARDI, G. **Using a Foundational Ontology for Reengineering a Software Process Ontology**. Journal of Information and Data Management, 2011.

BLOMQVIST, E.; GANGEMI, A.; PRESUTTI, V. **Experiments on Pattern-based Ontology Design.** Proceedings of the fifth international conference on Knowledge Capture - K-CAP'09, 2009.

BUSCHMANN, F.; HENNEY, K.; SCHMIDT, D. C. **Pattern Oriented Software Architecture Volume 5: On Patterns and Pattern Languages**. v. 5, 2007

FALBO, R. A., GUIZZARDI, G., GANGEMI, A., PRESUTTI, V. **Ontology Patterns: Clarifying Concepts and Terminology**. Proceedings of the 4th Workshop on Ontology Patterns (WOP2013), 2013.

GANGEMI, A., GUARINO, N., MASOLO, C., OLTRAMARI, A., & SCHNEIDER, L. **Sweetening ontologies with DOLCE**. In Knowledge engineering and knowledge management: Ontologies and the semantic Web (pp. 166-181). Springer Berlin Heidelberg, 2002.

GANGEMI, A.; PRESUTTI, V. **Ontology Design Patterns**. In: STAAB, S.; STUDER, R. (Eds.).Handbook on Ontologies. Springer, 2009. p. 221–243.

GRUBER, T. R. **A translation approach to portable ontology specifications**. Knowledge Acquisition, v. 5, n. 2, p. 199–220, 1993.

GUARINO, N.; GUIZZARDI, G. **"We Need to Discuss the Relationship": Revisiting Relationships as Modeling Constructs.** Advanced Information Systems Engineering. Springer, 2015

GUARINO, N.; OBERLE, D.; STAAB, S. **What Is an Ontology?**. Handbook on Ontologies, 2009.

GUARINO, N., **The ontological level: Revisiting 30 years of knowledge representation**. In Conceptual modeling: Foundations and applications (pp. 52-67). 2009.

GRAY, P. **Software engineering best practices: lessons from successful projects in the top companies**. 2010

GRAÇAS, A. P. **Suporte Automatizado Para Construção de Modelos Conceituais Bem Fundamentados**, Dissertação de Mestrado em Informática, Universidade Federal do Espírito Santo, 2010.

GUERSON, J. **Representing Dynamic Invariants in Ontologically Well-Founded Conceptual Models**. UFES, 2015.

GUERSON, J. et al. **OntoUML Lightweight Editor: A Model-Based Environment to Build, Evaluate and Implement Reference Ontologies.** Enterprise Distributed Object Computing Workshop (EDOCW), 2015 IEEE 19th International.

GUIZZARDI, G. **Ontological foundations for structural conceptual models**. Enschede: Telematica Instituut Fundamental Research Series, 2005.

GUIZZARDI, G., WAGNER, G., ALMEIDA, J.P.A., GUIZZARDI, R.S.S., **Towards Ontological Foundation for Conceptual Modeling: The Unified Foundational Ontology (UFO) Story**, Applied Ontology, Vol. 10, issues 3-4, IOS Press, 2015

GUIZZARDI, G., ZAMBORLINI, V., **Using a Trope-Based Foundational Ontology for Bridging different areas of concern in Ontology-Driven Conceptual Modeling**, Science of Computer Programming, Elsevier, 2014, DOI:10.1016/j.scico.2014.02.022.

GUIZZARDI, G. **Agent Roles, Qua Individuals and The Counting Problem**. Springer-Verlag, 2006.

GUIZZARDI, G. **On the Representation of Quantities and their Parts in Conceptual Modeling**, 6th International Conference on Formal Ontologies in Information Systems (FOIS 2010), Toronto, 2010.

GUIZZARDI, G., **Ontological Foundations for Conceptual Part-Whole Relations: The Case of Collectives and their Parts**, 23rd International Conference on Advanced Information System Engineering (CAiSE'11), London, UK.

GUIZZARDI, G. **Ontological meta-properties of derived object types.** Lecture Notes in Computer Science, 2012.

GUIZZARDI, G., WAGNER, G., FALBO, R.A., GUIZZARDI, R.S.S., ALMEIDA, J.P.A., **Towards Ontological Foundations for the Conceptual Modeling of Events**, 32nd International Conference on Conceptual Modeling (ER 2013), Hong Kong, 2013.

GUIZZARDI, G., FALBO, R. A., GUIZZARDI, R. S. S. **Grounding Software Domain Ontologies in the Unified Foundational Ontology (UFO): The case of the ODE Software Process Ontology**. In: XI Iberoamerican Workshop on Requirements Engineering and Software Environments (IDEAS'2008), 2008, Recife.

GUIZZARDI, G.; GRAÇAS, A. P.; GUIZZARDI, R. S. S. **Design patterns and inductive modeling rules to support the construction of ontologically well-founded conceptual models in OntoUML.** Lecture Notes in Business Information Processing, 2011.

GUIZZARDI, G. **Ontological patterns, anti-patterns and pattern languages for next-generation conceptual modeling**. Conceptual Modeling. Springer International Publishing, 2014. 13-27.

MAIDEN, N.; SUTCLIFFE, A. **Exploiting reusable specifications through analogy**. Communications of the ACM, 1992.

MYLOPOULOS, J., **Conceptual modeling and Telos**. In P. Loucopoulos & R. Zicari (eds.), Conceptual Modeling, Databases, and CASE (Chapter 2, pp. 49–68). Wiley, 1992.

MATTSSON, M.; BOSCH, J.; FAYAD, M. E. **Framework integration problems, causes, solutions**. Communications of the ACM, 1999.

NARDI, J. C. et al. **Towards a commitment-based reference ontology for services.** Enterprise Distributed Object Computing Conference (EDOC), 2013 17th IEEE International.

NOPPENS, O.; LIEBIG, T. **Ontology Patterns and Beyond-Towards a Universal Pattern Language.** WOP, 2009.

ODELL, J. J. **Advanced object-oriented analysis and design using UML**. SIGS reference library, 1998.

PEREIRA, D. C.; ALMEIDA, J. P. A. **Representing Organizational Structures in an Enterprise Architecture Language**. FOMI'2014 Formal Ontologies meet Industry, 2014.

POVEDA VILLALON, M.; SUÁREZ-FIGUEROA, M. C.; GÓMEZ-PÉREZ, A. **Reusing ontology design patterns in a context ontology network**. 2010.

REGINATO, C. C. **Propagação de Metapropriedades em Padrões de Derivação em OntoUML**. Universidade Federal do Espírito Santo, 2015.

RUY, F. B. et al. **Towards an Ontology Pattern Language for Harmonizing Software Process related ISO Standards**, 2015.

RUY, F. B. et al. Ontology Engineering by Combining Ontology Patterns. ER, 2015.

SCHERP, A. et al. **Designing core ontologies**. Applied Ontology, 2011.

SALES, T.P., GUIZZARDI, G., **Ontological Anti-Patterns: Empirically Uncovered Error-Prone Structures in Ontology-Driven Conceptual Models**, Data & Knowledge Engineering (DKE) Journal, 2015.

ZAMBORLINI, V. C. **Estudo de Alternativas de Mapeamento de Ontologias da Linguagem OntoUML Para OWL: Abordagens Para Representação de Informação Temporal**. Universidade Federal do Espírito Santo, 2011