UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
DEPARTAMENTO DE INFORMÁTICA
MESTRADO EM INFORMÁTICA

CÁSSIO CHAVES REGINATO

Propagação de Meta-propriedades em Padrões de Derivação em OntoUML

VITÓRIA – ES, Brasil

2015

CÁSSIO CHAVES REGINATO

Propagação de Meta-propriedades em Padrões de Derivação 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, Brazil

2015

Dados Internacionais de Catalogação-na-publicação (CIP)

(Biblioteca Setorial Tecnológica,

Universidade Federal do Espírito Santo, ES, Brasil)

Reginato, Cássio Chaves, 1986-

R335p Propagação de metapropriedades em padrões de derivação em OntoUML / Cássio Chaves Reginato. – 2015.

96 f. : il.

Orientador: Giancarlo Guizzardi.

Dissertação (Mestrado em Informática) – Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Ontologia. 2. Ontologias (Recuperação da informação). 3. Padrões de Derivação. I. Guizzardi, Giancarlo. II. Universidade Federal do Espírito Santo. Centro Tecnológico. III. Título

CDU: 004

DEDICATION

Dedico este trabalho a Srª Sandra Chaves por tudo que é e representa para mim.

ACKNOWLEDGEMENTS

Agradeço primeiramente a minha família: À minha mãe Sandra, meu pai de consideração e grande ídolo Ricardo Aragão, meus irmãos e amigos Arthur e Bruno. A meus queridos padrinhos Juan Banura e Leda Beatriz que sempre foram fundamentais na minha trajetória. Meus avós emprestados Brigone e Lorena que me ensinaram muito. E claro, não posso esquecer das Rita, a Marques e a Fabiane que foram e são exemplo para mim em todos os sentidos. Enfim, à todos os demais integrantes e agregados da família Chaves.

Aos mestres da minha graduação que despertaram em mim o desejo por conhecimento: Drª Karin Komati, Msc Hilario Seibel e Msc Fabiano Ruy (que foi também meu coorientador informal e parceiro neste trabalho). Dentre eles estão também os formadores da minha banca: Dr Mateus Costa e Dr Maxwell Monteiro a quem acima de todas as formalidades considero, no sentido restrito da palavra, meu mestre.

Finalmente, a todos os Professores e colegas do NEMO.

RESUMO

Muitas vezes, em modelagem conceitual, é preciso representar conceitos que surgem em função de inferências aplicadas sobre outros conceitos. Mais especificamente, Modelos conceituais ontológicos precisam representar tipos derivados considerando a consistência lógica e ontológica. Neste trabalho é apresentado padrões de tipos de objeto derivados em uma linguagem de modelagem conceitual que considera aspectos ontológicos, considerando as interações entre as meta-propriedades da linguagem e os vários tipos de derivação. Além disso, esses padrões são implementados em um editor de modelos conceituais ontológicos.

Palavras chave: Padrões de Derivação, Derivação, Ontologia, Ontologias, Tipos Derivados.

ABSTRACT

Quite often, in conceptual modeling, we need to represent a concept that can only exist in function of inferences applied in other elements. More specifically, Ontological Conceptual models need to represent derived types considering ontological consistence. Hence, in this work we discuss the implementation of derived types in ontology-driven conceptual modeling language regarding the interaction between the meta-properties of the language and the various types of derivation. Moreover, we have extended the OntoUML lightweight editor, providing derivation patterns to facilitate the building of ontological models.

Keywords: Derived Types, Derivation, Ontology, Ontologies, Derived Patterns

Lista de Fíguras

Lita de Tabelas

Sumário

1 INTRODUÇÃO

Este trabalho trata de aspectos de Modelagem Conceitual, atividade que tem por objetivo a construção de uma representação formal de um domínio para propósito de entendimento e comunicação (Mylopoulos & Schmidt 2010). Nas últimas décadas, modelagem conceitual tem sido aplicada como método de formalização de sistemas de informação de forma independente de implementação (Olivé 2007). Isso significa que durante o processo de análise de sistema, são desconsiderados aspectos puramente computacionais. Nesse contexto, destaca-se os aspectos que influenciam na produção de um modelo conceitual capaz de atender os objetivos que levaram a sua construção. De acordo com (Vojislav & Leon 2000), a qualidade de um modelo conceitual pode ser medida de acordo com sua completude e corretude sintática e semântica. Além desses fatores, Lindland et al. (1994) destacam a definição clara da linguagem, consistência e clareza na definição do escopo.

Como pode-se perceber, os fatores citados são impactados pela escolha de uma linguagem adequada e por sua correta aplicação. A linguagem apropriada para produzir uma especificação consistente deve fornecer um mapeamento coerente entre os níveis semântico, sintático e pragmático (Moody & van Hillegersberg 2009). Qualquer linguagem, inclusive linguagens de modelagem conceitual, podem ser específicas para um determinado domínio (i.e., linguagem específica de domínio) ou podem não se comprometer com um domínio específico (i.e., linguagem independente de domínio) (Guizzardi 2006b). Nesse trabalho, dá-se ênfase a linguagens independente de domínio. Uma Linguagem de modelagem conceitual independente de domínio deve fornecer construtos genéricos de forma a ser capaz de ser utilizada para descrever diversos domínios. Esses construtos fazem parte da sintaxe abstrata da linguagem, que define um metamodelo, que estabelece as regras, que devem ser respeitadas para que se possa ter construções consistentes nessa linguagem. Por exemplo, as linguagens que definem o conceito de herança como parte de sua sintaxe abstrata devem estabelecer uma regra em seu metamodelo, de forma que não seja possível que um tipo A herde propriedades de um outro tipo B que, por sua vez, herda propriedades desse mesmo tipo A. Ou seja, devemos definir que o conceito de herança é acíclico. A sintaxe concreta da linguagem, por sua vez, define símbolos léxicos de forma a representar construtos da sintaxe concreta.

A Unified Modeling Language (UML) (OMG 2010) é uma linguagem de modelagem independente de domínio, amplamente utilizada para produzir representações formais de sistemas. Embora UML seja, muitas vezes, utilizada para produzir modelos conceituais independente de plataforma (Evermann 2002), seu metamodelo é fortemente influenciado por conceitos de programação orientada a objetos. Em outras palavras, em UML, os construtos genéricos da linguagem (anteriormente citados como capazes de ser utilizados para descrever domínios variados) refletem os construtos de OO. Uma vez que as linguagens de modelagem voltadas para OO trabalham em um nível de abstração razoavelmente alto, é justificável sua utilização para construção de um modelo conceitual de domínio. Mesmo assim, por ser uma abordagem limitada por questões computacionais, tal abordagem acaba restringindo a capacidade de representação de um domínio a aquilo que uma linguagem de programação orientada a objetos consegue representar. O mesmo serve para linguagens de modelagem voltadas para construção de modelos de banco de dados como ER e ORM. Ou seja, essas linguagens acabam sendo impactadas por limitações da forma de representação de banco de dados relacionais.

Pode-se inferir, portanto, que tanto UML como as linguagens de modelagem de banco de dados, foram construídas para se projetar artefatos computacionais específicos, com base nas percepções humanas sobre os diferentes domínios. Olivé (1998) defende que, uma vez que linguagens de modelagem conceitual visam o entendimento e comunicação entre humanos, seus construtos devem ser livres de visões computacionais. Ou seja, mesmo que o objetivo da construção de um modelo conceitual seja a formalização de um sistema de informação a ser implementado em um paradigma OO, com banco de dados estruturado, argumenta-se que este modelo não deve considerar aspectos de implementação como diretriz para a correta representação do domínio. Um bom exemplo de uma restrição de implementação que não deve influenciar em um modelo conceitual é a limitação de algumas linguagens de OO com relação a herança múltipla. Conceitualmente, não há razão aparente para impedir-se que representemos que Pessoa seja um subtipo tanto de Ser Vivo como de Agente (considerando que nem todo agente é um ser vivo), o que configuraria um caso de herança múltipla.

Visando aumentar a expressividade de modelos conceituais construídos em UML, Guizzardi (2005) define uma linguagem que estende a sintaxe já consagrada de UML, sobrescrevendo seu metamodelo, baseando-se em meta-propriedades ontológicas oriundas de teorias da

metafísica e das ciências cognitivas. Essas meta-propriedades permitem aos modeladores construir modelos que tem uma maior chance de alcançar os critérios de qualidade mencionados, quando comparados com as demais linguagens que não aplicam tais distinções Guizzardi (2005). Nesse trabalho, o metamodelo de OntoUML foi construído de forma a refletir as distinções ontológicas da ontologia de fundamentação - *Unified Foundational Ontology* (UFO), também definida por Guizzardi (2005). Ontologias de fundamentação fornecem uma variedade de categorias ontológicas que montam uma fundamentação geral para ser utilizada na construção de ontologias de domínio. Uma vez que temos essas distinções como um perfil de UML (i.e., OntoUML), podemos, então, construir modelos conceituais com primitivas ontológicas, de forma a aumentar a expressividade dos modelos. OntoUML foi construída para possibilitar a construção de um modelo, com base em decisões que envolvem aspectos ontológicos e não computacionais.

Nesse trabalho, aborda-se um assunto tradicional em linguagens de modelagem conceitual aplicado a OntoUML. Trata-se da distinção entre tipos de objetos base e tipos de objetos derivados. Tipos não-derivados ou tipos base, são aqueles que podem ser diretamente instanciados de acordo com seus atributos. Mais especificamente, podemos definir que as instâncias de tipos base são determinadas por características intrínsecas (e.g., *José* é uma *Pessoa* porque possuí os atributos que o tornam uma *Pessoa*). Por outro lado, tipos derivados são instanciados com base em regras aplicadas em outros tipos do modelo (i.e., tipos derivados são definidos por propriedades extensionais). Ou seja, suas instâncias são determinadas indiretamente de acordo com a população de outros tipos (e.g., a população de *Solteiros* pode ser considerada todas as *pessoas* que não são *Casadas*, *Viúvas* ou *Divorciadas*). Olivé (1998) argumenta que tipos derivados podem ser ocultados de um modelo conceitual sem que esse perca expressividade. Por exemplo, se decidirmos não criar o tipo *Solteiro* em um modelo, ainda poderemos reconstruir suas instâncias com base na regra que o define. Ou seja, pode-se obter todas as instâncias que são *Pessoas* e que não estão *Casadas*, *Divorciadas* ou *Viúvas*. Logo, do ponto de vista lógico, tipos derivados podem ser considerados irrelevantes por levarem à criação de um tipo novo que não introduz nenhuma propriedade nova ao modelo. Por outro lado, em muitos casos, faz-se necessário a criação do tipo de objeto derivado para facilitar o entendimento do modelo ou, até mesmo, para representar tipos derivados que já são aceitos por uma comunidade (e.g., *Solteiro*).

Este trabalho investiga as formas de ocorrência dos tipos derivados especialmente para OntoUML, por meio da definição de padrões de derivação consistentes. A consistência desses padrões de derivação refere-se às formas de derivação válidas, que podem ser aplicadas em OntoUML, considerando as meta-propriedades ontológicas apoiadas pela linguagem. Em outras palavras, pode-se restringir a ocorrência de tipos derivados em OntoUML somente aos casos válidos em termos de padrões de derivação.

1.1 Motivação

Tipos derivados tem uma relevante tradição em linguagens de modelagem de banco de dados (Abiteboul & Hull 1987)(Hammer & Mc Leod 1981)(Halpin & Bloesch 1999). UML (OMG 2010), uma das linguagens mais utilizadas para construção de modelos de informação, também define construtos para relações e atributos derivados. Mesmo assim, as linguagens oferecem pouco ou quase nenhum suporte teórico para a aplicação desses tipos de forma adequada. Cabe ainda mencionar que nem mesmo a distinção sobre o que é e o que não é um tipo derivado se mostra consistente nas diversas linguagens que a aplicam. Uma lacuna a ser explorada, então, é a investigação das formas e padrões em que tipos derivados ocorrem.

Olivé (Olivé 1998) deu os primeiros passos para formalização desses tipos em UML provendo uma distinção entre tipos base, tipos híbridos e tipos derivados. Além disso, Olivé trata de casos especiais de derivação, que são formas de se definir um tipo com base em operações de conjuntos (i.e., união, exclusão, interseção etc.). Com base nos casos apresentados por Olivé, Guizzardi (2012) demonstrou que é possível derivar meta-propriedades ontológicas dos tipos de objeto para alguns casos especiais de derivação. Por exemplo, se tivermos um tipo que é derivado pela união de outros tipos (e.g., vamos supor que definimos que *Progenitor* é a união de *Pai* e *Mãe*) pode-se então definir sua categorização em OntoUML aplicando um padrão de derivação que determina as meta-propriedades ontológicas para esse tipo. Por exemplo, *Pai* e *Mãe* em OntoUML são categorizados como **Role**, ou seja, são papéis que uma *Pessoa* assume ao ter um *Filho*. Ao definir *Progenitor* como a união desses dois tipos, aplicamos um padrão que infere que essa união também será um **Role**. Percebemos, então, que as definições de padrões de tipos de objetos podem fornecer facilidades na construção de modelos conceituais em OntoUML.

Por aplicar conceitos ontológicos, OntoUML é comumente utilizada para construção de ontologias de referência. Uma Ontologia de referência é um modelo conceitual gerado como um artefato de engenharia que representa um modelo de consenso de uma determinada comunidade. Ontologias de Referência tratam de modelos para humanos interpretarem, sem qualquer aspecto computacional envolvido (Falbo 2011). Por outro lado, ontologias operacionais são artefatos computacionais (i.e., capazes de ser lidos por máquina), normalmente com menor expressividade em função das restrições impostas por aspectos computacionais. Ontologias operacionais (comumente chamadas de *lighweight ontologies*) são essenciais no contexto da Web Semântica, uma iniciativa para dotar a web de tecnologias semânticas, de modo que as páginas web possam ser interpretadas por software (Berners-Lee et al. 2001). Nesse contexto, ontologias provêem estruturas básicas de domínio descritas formalmente com lógica descritiva.

A *Web Ontology Language* (OWL) (W3C 2012), mais especificamente OWL-DL, é a linguagem de ontologia mais utilizada na Web Semântica, por prover uma estrutura taxonômica simples enriquecida com construtos de *Description Logic*. OWL-DL pode ser mapeada para diversas sintaxes como forma de expor as estruturas de conhecimento a serem consumidas por aplicações. Embora sejam chamadas de ontologias, um problema bastante comum em *lightweight ontologies* são seu baixo comprometimento ontológico (Guizzardi 2006a). Ou seja, encontra-se muitas inconsistências de nível conceitual em modelos da Web Semântica. Como forma de enriquecer a semântica dos modelos representados em OWL-DL, defende-se a construção de ontologias computacionais com base em ontologias de referência. Mais especificamente, pode-se, a partir de um modelo em OntoUML, gerar um modelo OWL-DL, por meio de diversos tipos de mapeamentos já realizados (Pedro Paulo F. Barcelos et al. 2013). Com base nesses mapeamentos, destaca-se a importância de se mapear os padrões de derivação em OntoUML para OWL-DL como forma de automatizar o processo de transformação de uma ontologia de referência para uma ontologia operacional. Ressalta-se também, a alta relevância dos padrões de derivação em OWL-DL, uma vez que essa linguagem é bastante aplicada para derivar informação através de raciocínio automático.

1.2 Objetivos da Pesquisa

Este trabalho tem como objetivo geral definir padrões de tipos de objetos derivados em OntoUML. Esse objetivo geral pode ser detalhado nos seguintes objetivos específicos:

(i) Determinar os métodos de propagação das meta-propriedades de OntoUML para tipos de objetos derivados;

(ii) Desenvolver suporte ferramental para utilização dos padrões de tipos de objetos derivados;

(iii) Implementar transformação automática dos padrões de tipos de objetos derivados em OntoUML para OWL-DL.

1.3 Método de Pesquisa

Este trabalho foi conduzido de acordo com os seguintes passos:

i) *Revisão da Literatura: Investigação das linguagens que consideram tipos derivados em sua especificação e a identificação dos casos gerais de derivação comumente aplicados.*

ii) *Criação dos Padrões de derivação em OntoUML: Propagação das meta-propriedades ontológicas de OntoUML para os casos gerais de derivação de forma a se gerar padrões consistentes para OntoUML.*

iii) *Suporte Ferramental:* desenvolvimento de suporte ferramental para aplicação dos padrões de derivação em OntoUML, incluindo transformação automática dos padrões de tipos de objetos derivados em OntoUML para OWL-DL.

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

1.4 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 texto é composto pelos seguintes capítulos:

- Capítulo 2 (Fundamentação Teórica): apresenta a base teórica para entendimento do trabalho.
- Capítulo 3 (Tipos Derivados e suas Representações): apresenta uma análise sintática e semântica de tipos de objetos derivados especialmente para UML.
- Capítulo 4 (Padrões de Derivação em OntoUML): apresenta os métodos de propagação das meta-propriedades ontológicas de forma a se produzir os padrões de tipos de objetos derivados.
- Capítulo 5 (Implementação): apresenta o suporte para tipos de objetos derivados em ferramenta de construção de ontologias em OntoUML bem como a transformação dos padrões para OWL-DL.
- Capítulo 6 (Considerações Finais e Trabalhos Futuros): apresenta considerações finais sobre as contribuições desta dissertação e delineia oportunidades futuras de pesquisa para dar continuidade a este trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a fundamentação teórica do trabalho, contemplando os principais assuntos relacionados ao tema. Primeiramente apresenta-se um subconjunto das meta-propriedades ontológicas de OntoUML que são aplicadas para criação dos padrões de tipos de objetos derivados. Considera-se então a forma com que essas meta-propriedades determinam as categorizações em OntoUML. Posteriormente são apresentados os casos especiais de derivação que determinam as possíveis formas de criação de tipos de objetos derivados. São apresentadas também as fases que envolvem a engenharia de ontologias para que se possa contextualizar as contribuições desse trabalho. Por fim é apresentado o mapeamento que serve de base para a transformação dos padrões de tipos de objeto derivados de OntoUML para uma linguagem de ontologia em nível operacional.

2.1 Meta-propriedades Ontológicas em OntoUML

Como mencionado anteriormente, OntoUML aplica em seu metamodelo várias meta-propriedades ontológicas (Guizzardi 2005). Este trabalho tem como foco três dessas metapropriedes, que são as que determinam as categorizações dos tipos de objeto em OntoUML (i.e., *Substantials*), sendo elas: Rigidez, Princípio de Identidade e Dependência. É importante ressaltar que serão, desde a seção 2.2.1, os tipos ontológicos encontrados em (Guizzardi 2005), assim, pressupõe-se conhecimento prévio do leitor quanto a esses tipos. Posteriormente, na seção 2.2.5, esses tipos são revisados.

2.1.1 Rigidez

Rigidez é uma propriedade modal de tipos. Um tipo é considerado rígido se sua instanciação for necessária a todos os indivíduos que o instanciam em algum mudo possível. Em outras palavras, a rigidez de um tipo determina se suas instâncias podem ou não deixar de ser suas instâncias sem deixar de existir. Tipos não-rígidos são aqueles cuja instanciação é contingente para pelo menos um dos indivíduos que o instanciam, e podem ser sub-classificados em tipos

semi-rígidos e tipos anti-rígidos conforme a abrangência da contingencialidade. Mais precisamente, temos as seguintes definições:

Para as definições envolvendo rigidez considere ***ext(T,m)*** a extensão de um tipo ***T*** em um mundo ***m*** pertencente ao conjunto ***M*** de todos os mundos possíveis.

Definição 1 (Rigidez): Um tipo ***T*** é rígido (marcado por ***R+***) se e somente se toda instância desse tipo é necessariamente uma instância desse tipo. Em outras palavras, a extensão do tipo ***T*** é invariante em relação ao conjunto de mundos possíveis, i.e., instâncias de ***T*** em qualquer mundo ***m*** são suas instâncias em todos mundos possíveis. Sendo assim, para todo **m,m' ∈ M** (sendo ***M*** o conjunto de todos os mundos possíveis) nós temos que ***ext(T,m)=ext(T,m')***.

Definição 2 (Anti-Rigidez): um tipo ***T*** é anti-rigido (marcado com ***R~***) se toda instância desse tipo pode, no sentido modal, deixar de ser uma instância desse tipo. Em outras palavras, existe um mundo $m \in M$ e uma instância x tal que $x \in ext(T,m)$ e em outro mundo $m' \in M$, nós temos que $m \in ext(T,m')$.

*Definição 3 (Não-Rigidez **R-**)*: Um tipo é não-rígido se ele não for rígido. Ou seja, se a rigidez não se aplicar a esse tipo.

*Definição 4 (Semi-Rigidez **R¬**)*: Um tipo é semi-rígido se ele é não-rígido mas não anti-rígido. Em outras palavras, um tipo semi-rígido é um tipo que é rígido para um subconjunto de suas instâncias e anti-rígido para um outro subconjunto de instâncias.

Para exemplificar cada possível variação quanto a rigidez, pode-se trabalhar alguns exemplos. O conceito de *Pessoa* pode ser considerado um tipo rígido uma vez que *Pessoas* nunca deixam de ser *Pessoas*. O conceito de *Estudante*, por sua vez, é anti-rígido, uma vez que sempre existirá um mundo em que um *Estudante* pode deixar de ser *Estudante*. Exemplos de tipos semi-rígidos são mais raros de ser encontrados. Um *Artigo de Luxo* pode ser um *Carro de Luxo* ou uma *Joia*. Um *Carro de Luxo*, com o passar do tempo, pode se tornar um *Carro Comum*, logo *Carro de Luxo* é um tipo anti-rígido. Já, uma *Jóia* pode ser vista como um tipo Rígido, uma vez que suas instâncias sempre serão *Jóias*. Sendo assim, *Item de Luxo* pode ser considerado um Mixin porque algumas de suas instâncias podem ser rígidas e outras anti-rígidas.

2.1.2 Princípio de Identidade

O Princípio de identidade é o critério que determina a identidade de indivíduos de um determinado tipo. A identidade é composta pelo conjunto de invariantes do mesmo. Por exemplo, ao se perder um pequeno pedaço de uma *Estátua de Argila*, provavelmente a estátua continuará sendo a mesma para quem a vê, porque seu princípio de identidade não foi perdido. Por outro lado, se essa *Estátua* for esmagada ao ponto de se tornar um *Pedaço de Barro* sem qualquer referência à peça original, seu princípio de identidade será perdido. Tipos podem ser classificados conforme a existência ou ausência de um princípio de identidade que se aplica universalmente às suas instâncias.

A metodologia de OntoClean (Guarino 2009) define duas meta-propriedades baseadas nessa característica dos tipos. Na primeira delas, os tipos que proveem identidade para suas instâncias e são marcados com *+O*. Os tipos que não proveem identidade para suas instâncias são marcados com -O. A segunda classifica os tipos que possuem critério de identidade de acordo com a existência ou não de um supertipo que também possui um critério de identidade. No primeiro caso a classe é *+I* e no segundo *-I*. Como exemplo, podemos considerar os tipos *Objeto Móvel, Pessoa* e *Mulher*: o primeiro é classificado como *-O* e –I pelo fato de possuir instâncias às quais se aplicam critérios de identidades distintos, e.g. tanto veículos quanto pessoas são instâncias de *Objeto Móvel*, embora atendam a critérios de identidade distintos. Por outro lado, *Pessoa* é classificada como *+O* e *-I*, pois todas as suas instâncias atendem ao mesmo critério de identidade e qualquer tipo que seja mais abrangente do que *Pessoa* possuirá alguma instância que não atende ao mesmo critério de identidade que instâncias de *Pessoa* atendem. Por fim, o conceito *Mulher* seria classificado como *-O* e *+I*, pois todas as suas instâncias atendem ao mesmo critério de identidade (o critério que se aplica a *Pessoa*) que, por sua vez, já está presente em um de seu supertipos (*Pessoa*).

2.1.3 Dependência

A visão completa do conceito de dependência é discutida em (Simons 1987). Em OntoUML, aplica-se a distinção entre dependência genérica e dependência existencial. A primeira delas, estabelece uma dependência entre tipos, e.g. o tipo *Estudante* é genericamente dependente de *Matrícula*, ou seja, todo *Estudante* pode cancelar uma *Matrícula* e ativar uma segunda, sem

deixar de ser *Estudante*. Em contrapartida, a dependência existencial é estabelecida no nível de instância e.g., *José* é existencialmente dependente de seu *Cérebro*. Um tipo é externamente dependente (marcado com D+) de outro tipo quando depende (seja genericamente ou existencialmente) de outro tipo. Um tipo é marcado como independente (marcado com D-) quando é possível existir instâncias desse tipo que não dependa externamente de outro tipo.

2.1.4 Combinando as Meta-propriedades Ontológicas

Quando se combina os possíveis valores para as meta-propriedades apresentadas, tem-se algumas restrições a ser consideradas para impedir construções inconsistentes. São elas:

Restrição 1 – Os Tipos provedores de identidade são obrigatoriamente rígidos e carregam princípio de identidade ***(+O*** $\rightarrow$ ***+I*** $\wedge$ ***R+***). Se um tipo provê identidade para todas as suas instâncias, então suas instâncias obrigatoriamente têm identidade. Caso ele fosse não rígido, quando uma de suas instâncias migrasse, sua identidade seria perdida por consequência. Logo, *um tipo provedor de identidade deve ser sempre rígido*.

Restrição 2 – Se um tipo carrega princípio de identidade e é anti-rígido então esse tipo herda seu princípio de identidade de outro tipo rígido provedor de identidade, seja direta ou indiretamente ($\forall$**x +I(x)** $\wedge$ **R-(x)** $\rightarrow$ $\exists$**!y +O(y)** $\wedge$ **subtipo-de(x,y**). Como demonstrado na Restrição 1, se um tipo provê identidade ele deve ser rígido. Sendo assim, *qualquer tipo não rígido que carrega um princípio de identidade deve herdar esse princípio de um tipo Rígido provedor de identidade.*

Restrição 3 – Um tipo semi-rígido obrigatoriamente não carrega princípio de identidade (R$\neg$ $\rightarrow$ -I). Tipos semi-rígidos tem uma parte rígida e outra anti-rígida. Sendo assim, pode-se facilitar o seu entendimento considerando que um tipo semi-rígido é sempre um tipo que generaliza pelo menos um tipo rígido e pelo menos um tipo anti-rígido (que não seja uma especialização do tipo rígido) para satisfazer sua condição. Sendo assim, *tipo semi-rígidos irão sempre representar abstrações de propriedades comuns a tipos com princípio de identidade diferente*.

Combinando as meta-propriedades, tem-se todos os casos válidos e inválidos considerando as restrições impostas. A **Tabela 2.1** apresenta todos os casos, marcando com a respectiva restrição os casos inválidos.

Tabela 2.1 - Combinação das Meta-propriedades

	Provedor de princípio de identidade (O)	**Herda Princípio de Identidade (I)**	**Rigidez**	**Dependência**	**Restrição**
1	+O	+I	R+	D-	
2			R+	D+	
3			R-	D+,D-	R1
4		-I	R+,R-	D+,D-	R1
5	-O	+I	R+	D+	
6				D-	
7			R~	D+	
8				D-	
9			R¬	D+, D-	R3
10		-I	R+	D-	
11				D+	
12			R~	D+	
13				D-	
14			R¬	D+	
15				D-	

2.1.5 Categorizações de OntoUML com Base nas Meta-propriedades Ontológicas

OntoUML é proposta como uma extensão de UML que incorpora, no metamodelo original de UML 2.0, axiomas que refletem as distinções da ontologia de fundamentação – *Unified Foundational Ontology* (UFO) (Guizzardi 2005). Este trabalho se concentra em um fragmento de UFO que trata dos **Substantial Universals**. De forma simples, **Substantial Universals** são objetos concretos como *Gatos, Pessoas, Casas, Carros, Mesas*. O conceito de **Substantial Universals** é o mais próximo em UFO para *tipos de objeto* em modelagem conceitual. As meta-propriedades ontológicas são aplicadas para eles, de forma a se obter um sistema de categorias de tipos de objeto em OntoUML. A primeira distinção aplicada é a separação entre **Sortal Universal** e **Mixin Universal** *(***Non-Sortal Universal***)*. **Sortal Universals** são marcados com +I enquanto **Non-Sortal Universals** são marcados com –I. Como afirmado anteriormente, tipos marcados com –I são obrigatoriamente abstrações de tipos marcados com +I. Ou seja, dois ou mais tipos que carregam princípios de identidade distintos (i.e., marcados com +I),

podem ser generalizados com base em propriedades comuns desses tipos, dando origem ao um tipo abstrato não provedor do princípio de identidade.

Os **Sortal Universals** podem ser rígidos ou anti-rígidos, não podendo ser semi-rígidos em função da Restrição 3. Dentre eles, somente **Sortal Universals** Rígidos podem prover identidade para suas instâncias em função da Restrição 1 (sendo, assim, marcados por O+). **Sortal Universals** Rígidos provedores de identidade são chamados de **Substance Sortals** *(***Substance Sortal***)*. Existem três tipos de **Substance Sortals** em OntoUML que são **Kinds, Quantities** e **Collectives** (veja na **Tabela 2.2**).

Tabela 2.2 - Substance Sortals

Estereótipo OntoUML	Definição
Kind	As instâncias de Kinds são complexos funcionais tais como (Pessoas, Cachorros e Árvore)
Quantity	Um Quantity define que suas instâncias são determinadas por quantidades (Ouro, Areia, Água)
Collective	Um Collective define que suas instâncias são coleções com uma estrutura uniforme (Time de Futebol, Equipe de Funcionários).

Os demais **Sortal Universals** existentes, surgem como especializações de tipos provedores de princípio de identidade. Logo, suas instâncias herdam o princípio de identidade provido pelo supertipo, uma vez que elas também são instâncias dele. Sendo assim, as categorizações apresentadas na Tabela (**Subkinds, Roles** e **Phases**) sempre surgem como subtipos diretos ou indiretos de **Substance Sortals.**

Tabela 2.3 – Sortal Universals que Herdam Princípio de Identidade

Estereótipo OntoUML	Definição
Subkind	Tipos rígidos que herdam o princípio de identidade de um Kind.
Role	Sortal anti-rígido relacionalmente dependente (Estudante, Trabalhador, Casado). Herda princípio de identidade de um Kind.
Phase	Sortal anti-rígido relacionalmente independente. Herda princípio de identidade de um Kind.

A Tabela 2.4 apresenta as diferentes categorizações para **Non-Sortal Universals** de acordo com o valor quanto a rigidez.

Tabela 2.4 – Non-Sortals

Estereótipo OntoUML	Definição
Category	É um Non-Sortal rígido que agrega propriedades de diferentes Substance Sortals por exemplo Móvel generaliza Cadeira e Mesa.
Role Mixin	Non-Sortal Anti-rígido
Mixin	Non-Sortal semi-rígido

Os estereótipos de OntoUML apresentados, cobrem os tipos de objeto possíveis. A **Figura 2.1** ilustra as distinções ontológicas de **Substantial Universals** que dão origem aos estereótipos de OntoUML destacados como folhas da árvore.

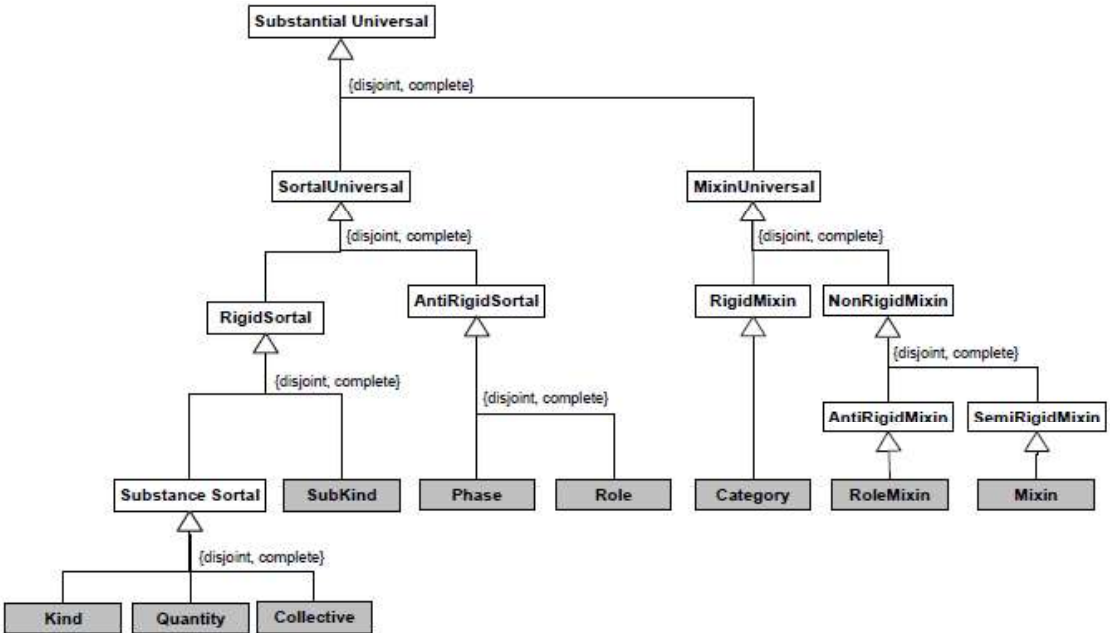

Figura 2.1- Hierarquia de Substantial Universals [fonte: (Guizzardi 2005)]

Para complementar o entendimento sobre as categorizações de OntoUML, a Tabela 2.5 apresenta, para cada estereótipo, seus respectivos valores para as meta-propriedades ontológicas. Além disso, a tabela também apresenta as generalizações e especializações válidas para cada categorização. Em outras palavras, para cada estereótipo marcado em um tipo ***T*** existe um conjunto válido de estereótipos que podem aparecer em um tipo ***T1*** que generaliza o tipo ***T*** e em um tipo ***T2*** que especializa o tipo ***T***. Por exemplo, na linha 5 um **Category** só pode ser generalizado por outro **Category** e pode ser especializado por qualquer outra categorização.

Tabela 2.5 – Meta-propriedades das Categorizações em OntoUML

ID	*Categorização OntoUML*	*Princípio de Identidade*	*Rigidez*	*Dependência Externa*	*Supertipos Válidos*	*Especializações Válidas*
1	Kind ∨ Collective ∨ Quantity	+I ∧ +O	R+	D-	5, 6	2,3,4
2	Subkind	+I ∧ -O	R+	D-	1, 2, 5, 6	2,3,4
3	Role	+I ∧ -O	R~	D+	Todos	3,4
4	Phase	+I ∧ -O	R~	D-	Todos	3,4
5	Category	-I ∧ -O	R+	D-	5,6	Todos
6	Mixin	-I ∧ -O	R ¬	D-	5, 6	Todos
7	Role Mixin	-I ∧ -O	R~	D+	5,6,7	3.7

O estereótipo **Relator** é a única categorização de OntoUML que não faz parte da hierarquia de **Substantial Universals,** mas que é fundamental para a construção os padrões apresentados neste trabalho. Um tipo é caracterizado como **Relator** quando ele é o responsável por conectar entidades. Por exemplo, uma *Pessoa* exerce o papel de *Funcionário* quando conectada com uma *Empresa* através de um *Contrato de Trabalho*. Logo, o *Contrato de Trabalho* pode ser visto como um **Relator**.

A **Figura 2.2** - Exemplo de Modelo de Casamento em OntoUML apresenta a representação de um possível modelo sobre casamento em OntoUML. Nota-se que uma *Pessoa* (**Kind**) pode assumir os papéis de *Marido* e *Esposa* (**Roles**) quando interconectados por um *Casamento* (**Relator**). Para esse modelo, um *Casamento* interconecta exatamente um *Marido* a uma *Esposa*.

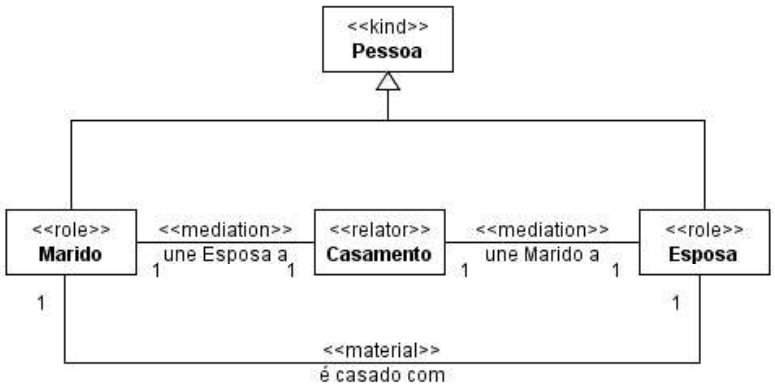

Figura 2.2 - Exemplo de Modelo de Casamento em OntoUML

2.2 Tipos Derivados

A classificação de um tipo como tipo base ou tipo derivado depende da forma como é descrito o princípio de aplicação, que, em suma, trata-se do princípio que define se algo é instância de uma classe ou não. Odell (1998) descreve os tipos derivados como tipos que definem seu princípio de aplicação em termos de um conjunto de predicados lógicos. Ou seja, é possível saber a população desse tipo aplicando regras de derivação de informação. Neste trabalho, segue-se a definição do Olivé (1998) para tipos derivados.

"A derivação de uma entidade ou relacionamento trata da forma pela qual o sistema de informação sabe a população de um tipo a qualquer momento. De acordo com esse aspecto, um tipo pode ser base, híbrido ou derivado".

Olivé (1998) propõem casos especiais de derivação como forma de se trabalhar os casos mais comuns de derivação que ocorrem em modelagem conceitual. São eles:

2.1. Derivação por União:

Um tipo ***T*** é derivado por união de $T_1,...,T_n$ $(n \geq 1)$ se a qualquer momento sua população é a união das populações de ***E1,...,En***.

2.2. Derivação por Exclusão:

Uma tipo T_1 é derivado por exclusão de $T_{2,...,}T_k$ se $T1,...,Tn$ são generalizados por um mesmo tipo T e se a população de T_1 é igual a população de T subtraindo-se a população de $T_2,...,T_n$.

A **Figura 2.3** apresenta exemplos de tipos que seriam derivados por união e por exclusão. A população dos *Seres Vivos* poderia ser calculada pela união das populações dos *Seres Unicelulares*, *Pluricelulares* e *Acelulares*. Sendo assim, o tipo *Ser Vivo* poderia ser considerado derivado por união.

A população de *Pessoas Desempregadas* poderia ser obtida através do complemento das *Pessoas Empregadas*.

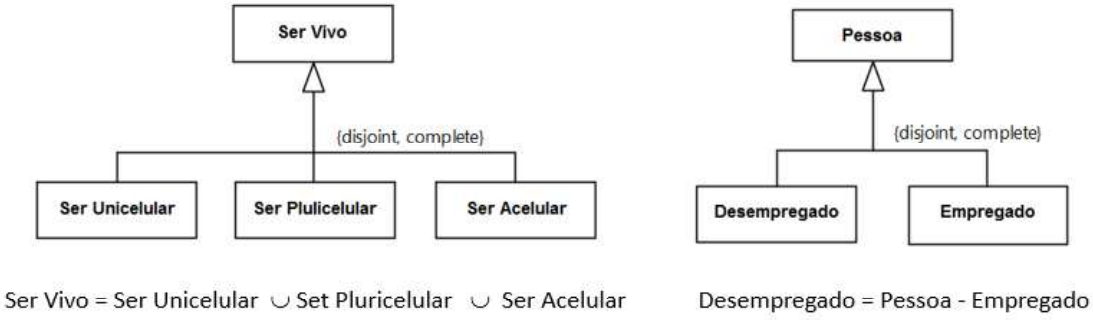

Figura 2.3 – Exemplo de Tipo Derivado por União e Tipo Derivado por Exclusão

2.3. Derivação por Especialização:

Um tipo $\boldsymbol{T}$ é derivado por especialização de $\boldsymbol{T_1}$ se todas as instâncias de $\boldsymbol{T}$ são também instâncias de $\boldsymbol{T_1}$ e se existe uma condição necessária e suficiente para determinar a população de $\boldsymbol{T}$ a qualquer momento. Ou seja, a população de $\boldsymbol{T}$ é um subconjunto da população de $\boldsymbol{T1}$ que atende a uma condição $\boldsymbol{C}$.

2.4. Derivação por Interseção

Um tipo $\boldsymbol{T}$ é derivado por interseção dos tipos $\boldsymbol{T_1,...,T_n}$ se a qualquer momento a sua extensão é a interseção dos tipos $\boldsymbol{T_1,...,T_n}$.

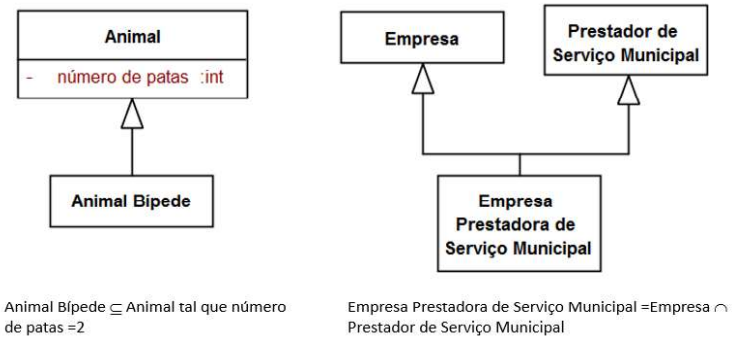

Figura 2.4 – Exemplos de Especialização e Interseção

A **Figura 2.4** apresenta exemplos de especialização e interseção. A população de *Animais Bípedes* pode ser a população de todos *Animais* tal que a condição *Número de Patas = 2* se aplica. Já uma *Empresa Prestadora de Serviço Municipal* deve ser tanto uma *Empresa* como um *Prestadora de Serviço Municipal*.

2.5. Derivação por Participação:

Considerando a relação ***R (p1:T1,...,pn:Tn)*** e um de seus participantes ***pi:Ti***. Assumindo que a participação de ***p1*** em ***R*** é parcial. Isso significa que é possível existir instâncias de ***Ti*** que participam em um dado relacionamento em ***R*** e outras não. ***T*** é uma entidade derivada por participação em ***R*** se sua população é definida como as instâncias de ***T1*** que participam em um relacionamento de ***R***. Uma derivação por participação é um tipo específico de derivação por especialização.

2.6. Derivação por Especialização Passada:

Um tipo ***T*** é derivado por especialização passada de T_1 se sua população a qualquer instante ***t*** for a soma das instâncias que já foram instâncias de T_1 no tempo ***t1*** anterior a ***t*** mas que não pertencem a T_1 em ***t***.

Como mostra a **Figura 2.5**, a população de *Gerentes* pode ser obtida como os *Funcionários* que gerenciam algum *Departamento* sendo então *Gerente* um tipo derivado por participação. Por fim, um *Ex-Atleta* é uma *Pessoa* que foi *Atleta* no passado mais não é atleta no mundo corrente, o que caracteriza *Ex-Atleta* como um exemplo de especialização passada.

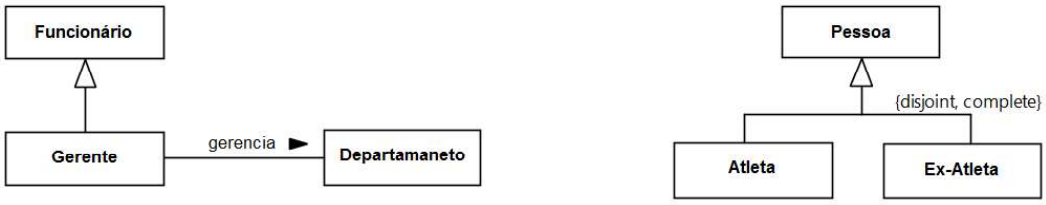

Figura 2.5 – Exemplos de Tipos Derivados por Participação e Especialização Passada

2.3 Fases da Engenharia de Ontologias

A linguagem OntoUML tem sido aplicada para a construção de ontologias de domínio, criando, para ela, um modelo conceitual. Sendo assim, os padrões de tipos de objeto derivados podem ser aplicados no contexto de engenharia de ontologias. Para entender como esses padrões podem ser úteis para construção de ontologias, é necessário conhecer as fases que envolvem a engenharia de ontologias.

Esse trabalho se baseia no método SABiO (Falbo 2011), que apresenta um processo de desenvolvimento de ontologias com base em fases genéricas. O método é constituído de 5 fases sequenciais: *Identificação do Propósito e Elicitação de Requisitos*, *Captura e Formalização da Ontologia*, *Design, Implementação* e *Testes*. De acordo com o propósito para o qual a ontologia está sendo desenvolvida, o processo pode ou não executar algumas das fases. Por exemplo, se o propósito é a construção de uma ontologia de referência, sem qualquer objetivo operacional, então o processo não necessitaria das fases de design e implementação. Como pode-se perceber, o método compreende as principais fases que envolve um processo completo de construção de ontologias, podendo ser adaptado para propósitos específicos. A **Figura 2.6** apresenta as fases do processo.

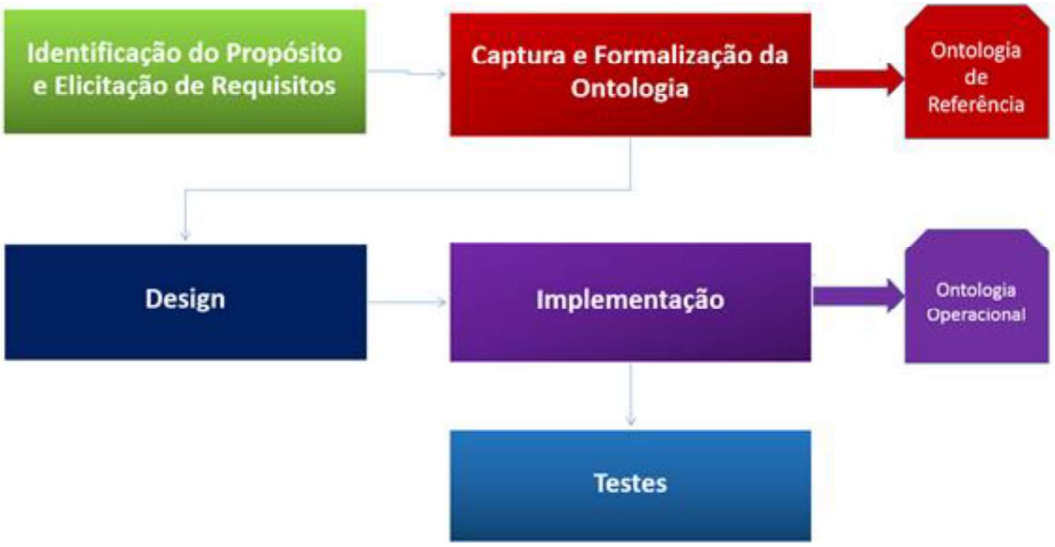

Figura 2.6 – Fases do Processo de Engenharia de Ontologias - SABiO

Padrões de tipos de objetos derivados estão relacionados a três das cinco fases apresentadas que são: Captura e Formalização da Ontologia, Design e Implementação. Por esse motivo, segue a explicação detalhada de cada uma dessas fases:

2.1.6 Captura e Formalização da Ontologia

O método SABiO divide essa fase em 4 atividades principais. A fase começa com a atividade de modelagem conceitual na qual os modeladores devem identificar os principais conceitos e relações do domínio construindo assim a estrutura do modelo. Cada conceito presente no modelo deve ser, então, definido no dicionário de termos, apresentando-se a fonte de onde tal definição foi extraída. Embora conceitos e relações sejam a base da ontologia, eles não são suficientes para capturar todo o domínio. Por isso, ainda nesta fase, é necessário definir os axiomas formais da ontologia, de forma a complementar o conhecimento presente no modelo conceitual.

Figura 2.7 – Atividades da fase de Captura e Formalização da Ontologia

O produto da fase de Captura e Formalização da Ontologia é uma Ontologia de Referência descrita formalmente como um modelo formal do conhecimento de um determinado domínio. O método SABiO sugere o uso de OntoUML como linguagem de representação da ontologia. Logo, como os padrões de tipos de objeto derivados em OntoUML podem ocorrer como parte da ontologia de domínio, por vezes produzindo axiomas complementares, pode-se concluir que esses padrões podem ocorrer na formalização desses modelos.

2.1.7 Design de Ontologias

A fase de *Design* compreende a transição de uma ontologia de referência para uma ontologia operacional a ser utilizada por aplicações computacionais. Sendo assim, um modelo conceitual que anteriormente tinha o propósito de representar o conhecimento formalmente para humanos, agora é adaptado para atender propósitos específicos de implementação. Aspectos não funcionais, como a linguagem de implementação a ser utilizada para codificar a ontologia, bem como aspectos arquiteturais, são considerados nessa fase. Nesta fase, no que se refere aos padrões de tipos de objeto derivados, pode-se considerar que são tomadas decisões quanto à representação dos padrões, de forma a atender os objetivos computacionais esperados.

2.1.8 Implementação de Ontologias

Nesta fase, a ontologia operacional é implementada em alguma linguagem de implementação computacional de acordo com as decisões de design previamente realizadas.

Nesse trabalho, trabalha-se especificamente com OWL-DL como linguagem de implementação de ontologias. Por essa razão, é apresentado a seguir um mapeamento de OntoUML para OWL-DL, visando automatizar o processo de transformação de uma ontologia de referência em uma ontologia operacional. Nessa transformação, são considerados aspectos de *design* que serão discutidos também por este trabalho no que diz respeito a padrões de tipos derivados.

2.4 Web Ontology language (OWL)

OWL (W3C 2016) é uma linguagem de representação de ontologias computacionais amplamente aplicada, em especial (mas não somente) para sistemas que funcionam na WWW. Por essa razão, este trabalho mapeia os padrões de tipos de objetos derivados de OntoUML para OWL.

OWL define diversas sintaxes e diferentes perfis para se adequar a diferentes propósitos. A linguagem permite a criação de taxonomias de Classes, Relações (i.e., Object Properties) e Atributos (i.e., Data Properties). O OWL-DL, perfil considerado nesse trabalho, aplica construtos de *Description Logic* como forma de aumentar o poder de formalismo de ontologias OWL. As propriedades de OWL definidas com base em Description Logic são bastante utilizadas para aumentar a capacidade de se produzir inferências automáticas através dos *reasoners*[1]. Os *reasoners* aplicam inferências em modelos OWL e produzem conhecimento novo de acordo com regras lógicas genéricas previamente definidas. Em OWL-DL pode-se aplicar quantificadores e operadores lógicos para a definição de Classes, Relações e Atributos. Pode-se, por exemplo, definir que a Classe Casado é equivalente a Classe Pessoa, para instâncias que possuem, pelo menos, um Casamento. OWL-DL garante que todas as conclusões lógicas aplicadas ao modelo são computáveis e finitas. Além disso, a especificação de OWL-DL possibilita a utilização de operadores de união e interseção entre Classes, facilitando, assim, a representação dos padrões de tipos de objeto derivados.

2.5 Uma Transformação Automática de OntoUML + OCL para OWL + SWRL

Existem diversos mapeamentos de OntoUML para OWL. Como exemplo, pode-se considerar os trabalhos de Zamborlini (2011) e Barcelos et al. (2013). Dentre esses dois, o mapeamento apresentado em (Barcelos et al. 2013) provê um processo de transformação de OntoUML para

OWL-DL que considera uma porção de parâmetros de *design*. Em outras palavras, de acordo com decisões de projeto, a transformação pode ser aplicada, considerando ou não propriedades específicas da linguagem.

Para demonstrar o método de mapeamento da transformação, a **Figura 2.8** apresenta um fragmento de modelo de OntoUML sendo mapeado para OWL-DL. Nesse fragmento, o conceito *Pessoa* é particionado entre *Homem* e *Mulher*, sendo o *Generalization Set (GS)* referente a esse particionamento, disjunto e completo. Na parte B, demonstra-se a hierarquia de classes em OWL-DL, que respeita a estrutura de OntoUML, enquanto a parte C e D representa respectivamente a definição de *Pessoa* e *Homem*. A definição de *Pessoa* como equivalente a *Homem* ou *Mulher* se dá devido ao GS ser definido como completo. Ou seja, toda instância de *Pessoa* obrigatoriamente é instância de um de seus subtipos. Como podemos ver na parte D, *Homem* é definido como disjunto de *Mulher* assim como definido em OntoUML.

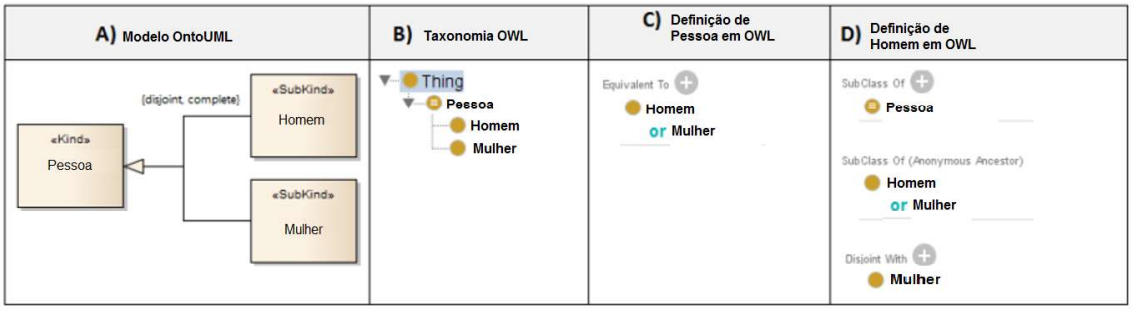

Figura 2.8 – Mapeamento de modelo OntoUML para OWL-DL

3 TIPOS DE OBJETO DERIVADOS EM MODELAGEM CONCEITUAL

3.1. Introdução

Este capítulo apresenta uma análise da aplicação de tipos de objetos derivados em linguagens de modelagem conceitual. O capítulo traz, ainda, algumas considerações sobre a forma de representação dos tipos derivados em UML, previamente proposta por Olivé (1998).

IFO (Abiteboul & Hull 1987) e SDM (Hammer & Mc Leod 1981) foram as primeiras linguagens de modelagem conceitual, mais especificamente linguagens de modelagem de banco de dados, a levarem em conta tipos derivados em seu metamodelo. Mais tarde, linguagens de modelagem conceitual amplamente difundidas, tais como UML (OMG 2014) e ORM (Halpin 2006), também os implementaram. As seções 3.2 a 3.5 apresentam as interpretações para tipos derivados em cada uma das linguagens.

3.2. Tipos Derivados em IFO

IFO é uma linguagem da década de 80, que tinha como objetivo introduzir alguns conceitos semânticos em modelagem de banco de dados. Dentre as sugestões de melhorias na semântica dos modelos, está a distinção entre tipos atômicos e tipos derivados.

Embora IFO trate de tipos derivados em sua especificação, nenhuma definição do que é um tipo derivado é provida. Sendo assim, a compreensão da interpretação de IFO para tipos derivados se dá pelas formas de ocorrência desses tipos. São definidas duas formas de derivação: Coleções e Agregações.

A **Figura 3.1** apresenta um exemplo de cada caso de derivação proposto em IFO. No primeiro, temos o caso de um coletivo extencional, que seria o conceito de *Turma* sendo derivado do conjunto de *Estudantes*, que devem estar matriculados no mesmo *Curso*. E no segundo, uma agregação *Barco Motorizado* sendo derivado das suas partes *Motor* e *Casco*.

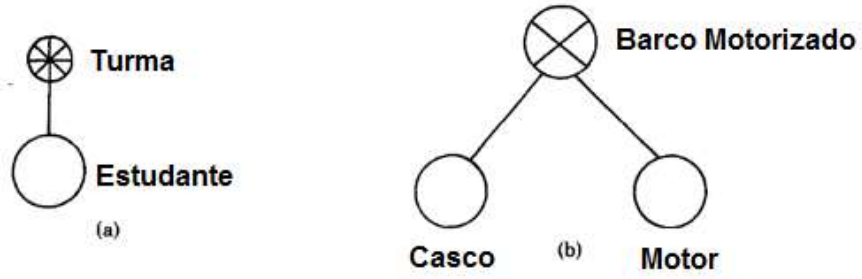

Figura 3.1 -Exemplos de tipos derivados em IFO

Uma Coleção é equivalente a ideia de *Collective* de OntoUML. A especificação de Coleções em IFO são descritas como agrupamento, que corresponde a um conjunto finito de objetos de uma dada estrutura específica (e.g., *Navio* e *Comboio*). Como exemplo, considere os *Comboios*, que são compostos unicamente por *Navios* agrupados por algum critério de unicidade, tais como a localização ou o fato de realizarem uma determinada função.

Já uma agregação em IFO corresponde ao que em UFO é chamado de um complexo funcional. Um complexo funcional, ao contrário de um coletivo, possui uma estrutura complexa e heterogênea. Por exemplo, um *Ser Humano* é composto de um complexo de estruturas distintas como *Células*, *Braços*, *Coração*, *Sangue*, etc.

Analisando os tipos de derivação de IFO, pode-se concluir que a linguagem define estruturas padrões de derivação arbitrários para fins específicos não claramente especificados. Para o caso do coletivo, se o critério de unicidade do coletivo for expresso explicitamente em termos de uma regra lógica que permita conhecer a população desse coletivo, então teríamos de fato um tipo derivado, de acordo com a conceituação proposta de derivação desse trabalho. Por outro lado, não se pode restringir os coletivos a critérios extensionais, uma vez que, como apontado por Guizzardi (2005), existem Coletivos intensionais, definidos por propriedades que não são somente o critério de unicidade. Em outras palavras, coletivos intensionais possuem características próprias que determinam seu princípio de aplicação.

Os tipos derivados por agregação em IFO não deveriam ser considerados derivados, segundo a definição de tipos derivados defendida neste trabalho. Primeiramente, porque complexos funcionais não podem ser definidos como a soma de suas partes, ou seja, ele não obedece um critério de identidade extencional e, como tal, pode perder alguma de suas partes não essenciais

e continuar existindo (e.g., uma *Pessoa* pode perder uma *Unha* e continuar sendo *Pessoa*). Além disso, o arranjo das propriedades essenciais mínimas necessárias para caracterizar um complexo funcional não implica em uma instância desse tipo. Por exemplo, a soma das partes de um *Ser Humano*, ou seja, um *Coração*, um *Cérebro* e as demais partes essenciais reunidas não são suficientes para caracterizar um *Humano*.

3.3. Tipos Derivados em SDM

SDM, assim como IFO, foi projetada para prover semântica formal para a estruturação de modelos de banco de dados. Como descrito em sua especificação, um modelo desenvolvido com SDM deve ser capaz de descrever um banco de dados em termos de seus tipos de entidades que existem em um ambiente de aplicação, as classificações e agrupamentos dessas entidades, e as interconexões estruturais entre elas (Hammer & Mc Leod 1981).

SDM distingue tipos base e tipos não base. Segundo a definição dada na especificação de SDM, um tipo não base equivale a um tipo derivado. "*Uma classe não base é aquela que não tem existência independente; além disso, ela é definida em termos de uma ou mais outras classes*" (Hammer & Mc Leod 1981, p. 357). Mais especificamente, SDM trabalha com dois casos de tipos derivados: Subclasse e *High-Order*. Uma subclasse derivada é aquela que estabelece explicitamente a condição para a especialização de uma Classe em termos de uma Subclasse. Ou seja, sua definição é equivalente à definição de tipo derivado por especialização dada por Olivé. *High-Order* é um tipo de segunda ou maior ordem, tal que suas instâncias são tipos da ordem inferior. Ou seja, uma instância de um tipo de segunda ordem é um tipo de tipo de primeira ordem. Por exemplo, o conceito de *Espécie Biológica* pode ser considerado um tipo de segunda ordem porque suas instâncias (e.g., *canis familiares* conhecido como *Cão*) são tipos de primeira ordem, que possuem instâncias que são *Cães* como um possível *Cão* chamado *Lassie*. Não há, na especificação de SDM, indícios do porquê *High-Orders* são considerados tipos derivados, de acordo com a própria definição de tipo derivado provida. De qualquer forma, *High-Orders* estão fora do escopo desse trabalho.

3.4. Tipos Derivados em ORM

Considerando os casos de derivação de ORM, o único que trata de um caso especial de derivação de tipos de objeto é a derivação de Subtipos. Halpin (2007) apresenta três razões para uma especialização de tipos: (1) para indicar que algumas propriedades são específicas de um determinado subtipo (e.g. a situação da *Próstata* é registrado apenas para *Pacientes Homens*); (2) para agrupar propriedades comuns de tipos (e.g. tanto *Pessoa Física* como *Jurídica* possuem um *Endereço*); (3) para representar uma taxonomia (e.g. Um *Paciente* pode ser *Homem* ou *Mulher*).

Inicialmente, em ORM, todos os subtipos eram derivados, ou seja, para toda especialização, era necessário informar as condições em termos de regras. Por exemplo, um *Paciente* é determinado como *Masculino* ou *Feminino* de acordo com o valor do seu gênero, como exibido na **Figura 3.2**. Posteriormente, ORM liberou os tipos oriundos de especialização a serem base, semi-derivados ou derivados. Ou seja, podemos ter um particionamento de um tipo, sem necessariamente especificar a condição de especialização.

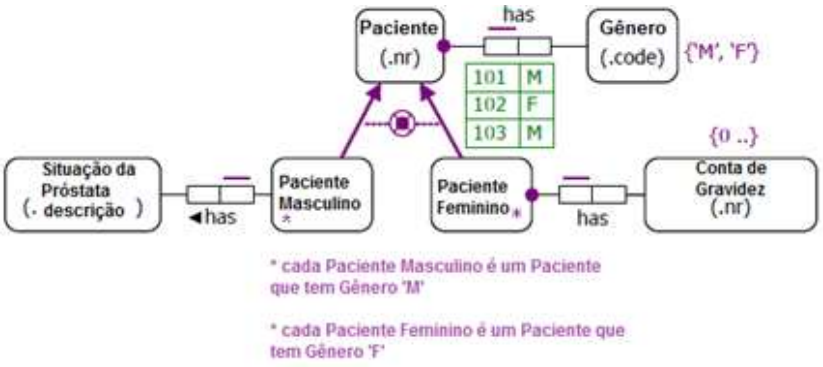

Figura 3.2 - Exemplo de tipo derivado por especialização em ORM

3.5. Representação dos Casos Especiais de Tipos Derivados em UML

A linguagem UML define construtos somente para atributos e relacionamentos derivados. A especificação de UML (OMG 2015) não faz menção a tipos de objetos derivados. Para suprir essa necessidade, Olivé (1998) sugeriu a representação dos casos especiais de tipos de objeto derivados em UML. A seguir, apresenta-se a proposta de Olivé, bem como algumas considerações importantes para subsidiar o trabalho de pesquisa desta dissertação.

3.5.1. Representação de Tipo Derivado por União um UML

Olivé argumenta que um tipo derivado por união é equivalente a uma classe abstrata. Uma classe abstrata, em UML, é definida como uma classe que não pode ser diretamente instanciada. Em outras palavras, uma instância de uma classe abstrata será obrigatoriamente instância de exatamente uma de suas subclasses concretas. A especificação de UML (Anon 2014) estabelece que todos *Generalization Sets* marcados como *complete* (i.e. todas instâncias do tipo genérico obrigatoriamente são instâncias dos tipos específicos) devem ter uma classe abstrata como supertipo. Essa afirmativa leva a um questionamento sobre a intepretação do conceito de instanciação em UML. Para responder a esse questionamento, pode-se supor que *Pessoa* seja definida como uma classe abstrata que define um *generalization set* marcado como *complete* para os subtipos *Homem* e *Mulher*. Quando se interpreta os exemplos providos de classes abstratas na especificação de UML ("OMG Unified Modeling LanguageTM Infrastructure," 2014), nota-se que a instanciação reflete as implementações de linguagens de programação orientadas a objeto. Em outras palavras, a instanciação das classes ocorre sempre do tipo mais específico para o mais genérico. Dessa forma, a classe Pessoa seria mesmo abstrata, uma vez que as instâncias de Pessoa seriam primeiro instanciadas como Homem ou Mulher.

Abstraindo de métodos de classificação computacionais, pode-se utilizar outros critérios para uma instanciação ser considerada direta, por exemplo a partir do princípio de identidade. Pode-se considerar, assim, que uma instanciação direta ocorre nos tipos provedores de princípio de identidade (para uma discussão detalhada sobre esse princípio, ver capítulo 2, em especial a seção 2.2.2). Nesse caso, tomando o exemplo mencionado no parágrafo anterior, uma vez que a *Pessoa* é um tipo provedor do princípio de identidade, então sua instanciação é direta e a classificação das instâncias de *Pessoa* como *Homem* ou *Mulher* é subsequente e derivada de suas características intrínsecas. Nesse caso, embora o *Generalization Set* que determina o gênero da pessoa seja marcado como *complete*, *Pessoa* não seria um tipo abstrato. Indo mais adiante, pode-se então afirmar que, segundo essa ideia, tipos provedores de princípio de identidade em OntoUML (Kinds, Colletives e Quantities em OntoUML) nunca poderiam ser abstratos. Este trabalho adota esta noção em relação à instanciação direto, mantendo, assim, a premissa de que tipos derivados por união são equivalentes a tipos abstratos, porém eliminando a regra estabelecida anteriormente para todo *Generalization Set* marcado como complete. Em

outras palavras, admite-se, neste trabalho, que haja um *Generalization Set* marcado como *complete* e que não tenha um supertipo abstrato.

3.5.2. Representação de Tipo Derivado por Exclusão em UML

No caso de tipos derivados por exclusão, Olivé propõe o uso de regras OCL para expressar a regra de exclusão, tal como mostra a regra exibida na **Figura 3.3**, sendo aplicada para o exemplo em que *Desempregado* seria o complemento de *Empregado*.

Figura 3.3 - Regra de Exclusão Proposta por Olivé

A tentativa da regra proposta é sobrescrever o método "allInstances()" de OCL, que retorna todas as instâncias de um determinado tipo. Sobrescrevendo esse método, a qualquer momento que for necessário saber as instâncias de *Desempregado*, será chamado o método *AllInstance* sobrescrito, de forma que a população de *Desempregado* sempre será o complemento de *Empregado*. O problema dessa solução é que OCL em sua especificação não permite a sobrescrita de métodos pré-definidos da linguagem e, em função disso, as implementações de OCL que são utilizadas neste trabalho também não aplicam esse recurso. Para resolver essa questão, propõe-se uma nova regra OCL, em termos de uma invariante, que utiliza uma implicação lógica (*implies*) para representar a exclusão. Segue a regra proposta:

```
Context Pessoa
inv: not self.self.oclIsKindOf(_'Empregado') implies
self.oclIsKindOf(_Desempregado')
```

A regra exibida acima define que se uma *Pessoa* não for classificada como *Empregado*, então será classificada como *Desempregado*.

3.5.3. Representação de Tipo Derivado por Interseção em UML

Tipos derivados por interseção são definidos por uma interseção total. Sendo C a interseção total de A e B, então todos os elementos que pertencem tanto a A quanto a B obrigatoriamente pertencem a C. Uma classe UML que especializa duas outras classes apenas representa uma interseção parcial. Para tornar tal classe uma interseção total, é preciso, novamente, definir uma regra OCL, como demonstrado na **Figura 3.4**.

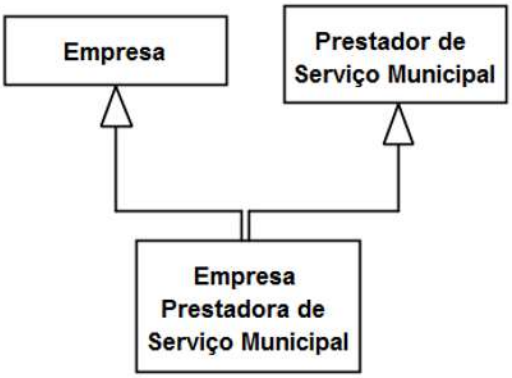

Figura 3.4 - Interseção Total em UML

3.5.4. Representação de Tipo Derivado por Especialização em UML

Assim como ORM, especializações em UML podem ou não definir explicitamente a condição de especialização. Segundo Olivé, uma condição definida em OCL para a especialização gera um tipo derivado por especialização. Ao contrário das regras de interseção e exclusão, a regra de especialização não segue um padrão pré-estabelecido. Isso ocorre porque as condições de especialização podem ser realizadas de incontáveis maneiras sendo aplicada em um ou diversos atributos de uma superclasse. Sendo assim, a regra de derivação deve sempre ser construída como uma invariante, como demonstrado na **Figura 3.5**.

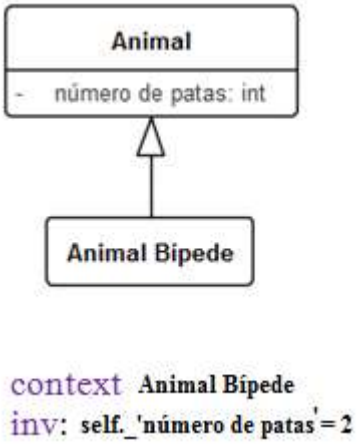

Figura 3.5 - Exemplo de Derivação por Especialização em UML

-

3.5.5. Representação de Tipo Derivado por Participação em UML

Um tipo derivado por participação pode ser representado através de uma relação não mandatória, tal que as instâncias que participam da relação se tornam a população do tipo derivado. Por exemplo, dentre os *Funcionários* de um *Departamento*, aqueles que se relacionam com *Departamento* através da relação de *gerência*, são classificados como *Gerentes*. Nesse caso, é necessário expressar uma regra OCL que classifica um *Funcionário* como *Gerente*, caso tal funcionário gerencie algum *Departamento*, tal como mostrado na

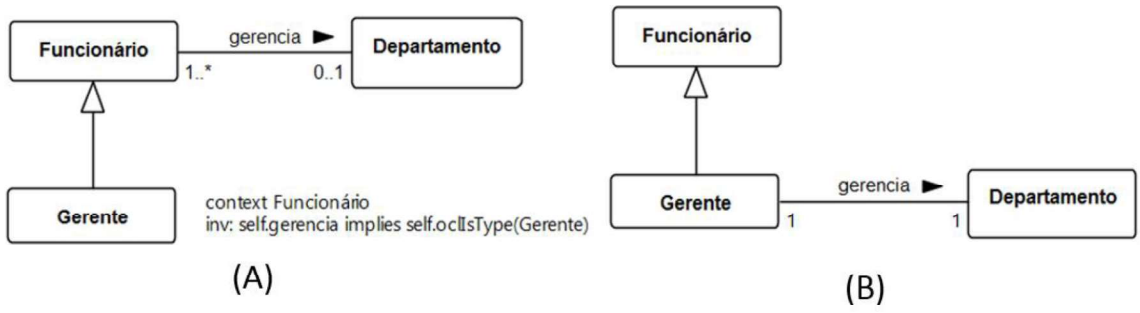

Figura 3.6 - Opções de Representação para Tipo Derivado por Participação**Figura 3.6** (A). Outra maneira de representar um tipo derivado por participação seria estabelecendo como domínio da relação o tipo derivado. Dessa forma a relação se tornaria mandatória. Seguindo com o exemplo, um *Gerente* seria então alguém que obrigatoriamente *gerencia* um *Departamento* como mostra a parte direita da

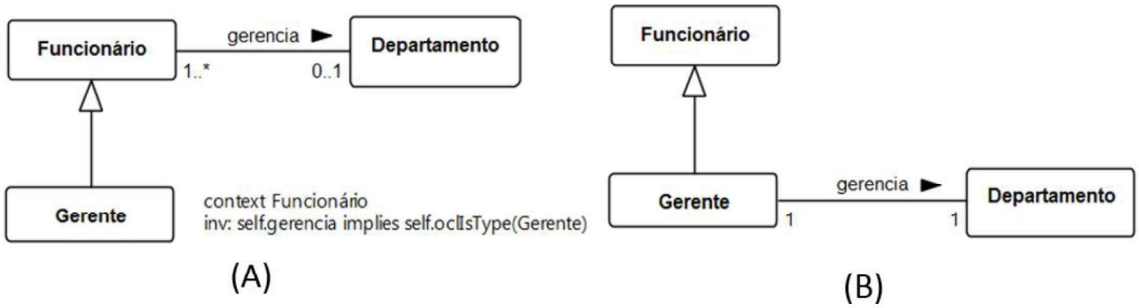

Figura 3.6 - Opções de Representação para Tipo Derivado por Participação (B). Neste trabalho, para representação de tipo derivado por participação, adota-se a segunda opção, lidando-se, portanto, somente com relações mandatórias que descrevem propriedades não opcionais dos tipos.

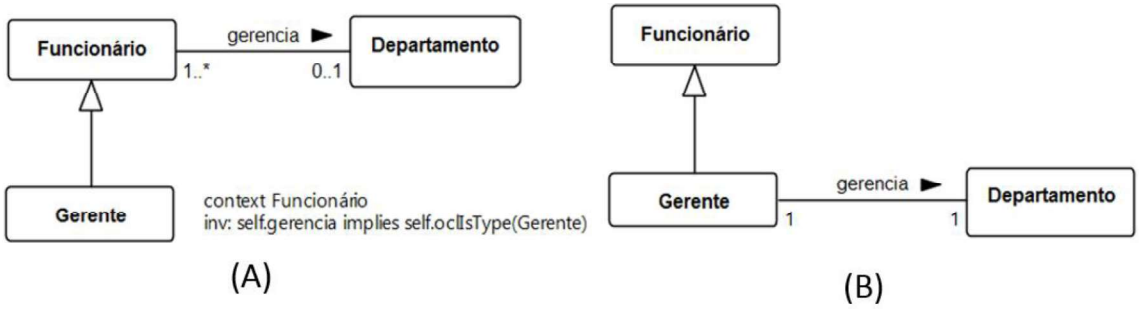

Figura 3.6 - Opções de Representação para Tipo Derivado por Participação

4 PADRÕES DE TIPOS DE OBJETOS DERIVADOS EM ONTOUML

4.1 Introdução

Nesse capítulo apresenta-se a principal contribuição deste trabalho que é a criação dos padrões de tipos de objetos derivados em OntoUML. Inicialmente apresenta-se as definições gerais da abordagem. Posteriormente apresenta-se o processo para criação dos padrões de tipos de objetos derivados em OntoUML. Por fim, instancia-se o processo para cada caso de derivação.

4.2 Definições Gerais

4.2.1 Derivation Set

Para a criação dos padrões de tipo de objetos derivados define-se o que é chamado de *Derivation Set* (DS). O DS representa o conjunto de elementos no qual um padrão de derivação é aplicado. Em outras palavras, a população do tipo de um objeto derivado é obtida de acordo com a população dos tipos que compõem o *Derivation Set.* Por exemplo, os tipos Mãe e Pai podem vir a formar um DS para a criação de um tipo de objeto derivado por união desse DS que nesse caso seria Progenitor. Logo, para aplicar um padrão de tipos de objeto derivado em OntoUML precisa-se definir um DS e aplicar algum tipo de derivação. Segue a definição:

DS é um subconjunto dos Tipos de Objeto de um Modelo,

Tipos de Objeto de um DS ≥ 1.

Além disso, define-se uma restrição geral para qualquer DS:

4.2.1.1 Restrição Geral 1

Um tipo não pode compor um DS juntamente com um de seus subtipos, seja ele direto ou indireto.

4.2.1.2 Notações de ocorrências de Propriedades e Categorizações OntoUML em um DS:

Para facilitar o entendimento sobre a criação dos padrões, utiliza-se uma notação para determinar o número de ocorrência de um valor de rigidez ou de uma categorização OntoUML em um determinado DS.

Em relação a rigidez pode-se definir que:

C_{R+}: é o conjunto de tipos de objeto rígidos de um modelo,

C_{R-}: é o conjunto de tipos de objeto não rígidos de um modelo,

$C_{R\sim}$: é o conjunto de tipos de objeto anti-rígidos de um modelo,

$C_{R\neg}$: é o conjunto de tipos de objeto semi-rígidos de um modelo.

Em relação as categorizações OntoUML pode-se definir que:

C_{Kind}: é o conjunto dos tipos de objeto de um modelo categorizados como Kind. O mesmo serve para todos as demais categorizações de tipos de objeto de OntoUML. Por exemplo, C_{Role} é o conjunto dos tipos de objeto de um modelo categorizados como Role.

Considerando essas definições pode-se determinar a ocorrência de valores de rigidez e de categorizações OntoUML em um DS seguindo os padrões a seguir:

$DS \subseteq C_{R+}$: O DS é formado somente por tipos rígidos,

$DS \cap C_{R+} \neq \varnothing$: O DS tem pelo menos um tipo rígido,

$DS \cap C_{Phase} = \varnothing$: O DS não tem nenhum Phase.

$DS \cap C_{Role} \neq \varnothing$, $DS \cap C_{Phase} \neq \varnothing, DS \subseteq (C_{Phase} \cup C_{Role})$: O DS é formado por pelo menos um Phase, um Role e somente por Phases e Roles.

4.2.2 Outras Definições

4.2.2.1 Extensão de um Tipo de Objeto

Utiliza-se uma notação de modalidade para lidar com a rigidez dos tipos (uma vez que a propriedade de rigidez trata de aspectos modais). Para expressar a extensão de um determinado tipo T em dado mundo m utiliza-se a notação ext(T,m).

4.2.2.2 Restrição Geral 2

A restrição 2 define que tipos derivados não podem ser categorizados como *Substance Sortals (Kinds, Collectives, Quantities)*. Isso porque os tipos categorizados como *Substance Sortals* provem princípio de identidade para suas instâncias. Portanto, do ponto de vista ontológico, esses tipos não deveriam ocorrer como um tipo de objeto derivado de outros tipos de objeto uma vez que esse tipo tem propriedades essenciais que determinam o princípio de identidade de suas instâncias.

4.3 Processo de Construção dos Padrões de Tipos Derivados em OntoUML

Para a construção dos padrões de tipos de objetos derivados em OntoUML, determina-se um processo a ser aplicado de forma que seja possível cobrir todos os casos possíveis de tipos de objetos derivados na linguagem. Considerando as meta-propriedades ontológicas da linguagem, essa derivação pode ser válida ou inválida. Sendo assim, o primeiro passo é definir quais são os *Derivation Sets* válidos para cada caso de derivação. Em outras palavras precisa-se definir quais são as configurações possíveis e válidas de estereótipos OntoUML que podem aparecer juntos em um determinado DS aplicado a um tipo específico de Derivação. Por exemplo, se tivermos dois *Substance Sortals* em um DS e aplicarmos uma derivação por união, teremos uma derivação válida. Porém uma derivação por interseção aplicada nesse mesmo DS, seria inválida uma vez que as instâncias desse tipo teriam dois princípios de identidade distintos. Por fim, define-se então para cada caso válido de derivação o seu respectivo valor em termos de estereótipos OntoUML.

Para cobrir todos os casos de tipos objeto derivados, o processo aplica uma sequência de etapas para cobrir as meta-propriedades de OntoUML para cada caso de derivação (i.e. União, Exclusão, Especialização, Interseção, Participação e Especialização Passada):

Etapa 1: A primeira etapa é verificar as combinações de valores possíveis quanto a rigidez dos tipos de objeto do DS.

Etapa 2: Para cada combinação de valores de rigidez válido do DS determina-se os valores de rigidez válidos para o tipo de objeto derivado. Cada configuração válida quanto a rigidez é chamada de um Caso de Derivação.

Etapa 3: Posteriormente são considerados as combinações de estereótipos OntoUML possíveis para cada caso considerados os valores quanto ao princípio de identidade e dependência dos tipos.

Etapa 4: Como resultado tem-se a geração dos Padrões de tipos de Objeto Derivado em OntoUML.

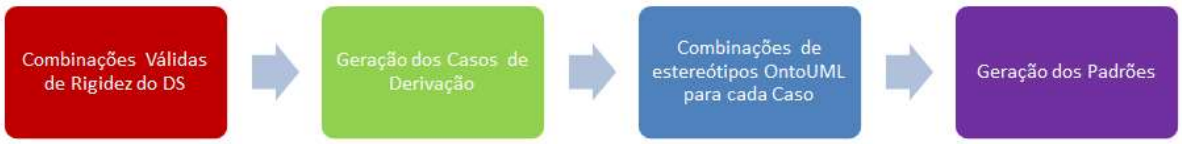

Figura 4.1 - Processo de Criação dos Padrões de Tipos de Objetos Derivados

4.4 Padrões de Tipos Derivados por União em OntoUML

4.4.1 Valores de Rigidez em Padrões de Tipos Derivados por União

Tipos derivados por união podem ser aplicados em qualquer combinação de valores de Rigidez. Sendo assim, separa-se três casos possíveis que compreendem todas as combinações possíveis em um DS. Para todos os casos supõe-se um tipo A sendo derivado por união de um DS. Ainda tem-se dois conjuntos nomeados SC_R (contendo todos os subtipos rígidos de A) e o subconjunto SC_{AR} (contento todos os subtipos anti-rígidos de A). Ambos subconjuntos podem ser vazios.

Caso 1 : Todos Tipos do DS Rígidos ($DS \subseteq C_{R+}$)

Se todos os tipos de um DS forem rígidos então o tipo derivado por união será rígido também como mostra a **Figura 4.2**.

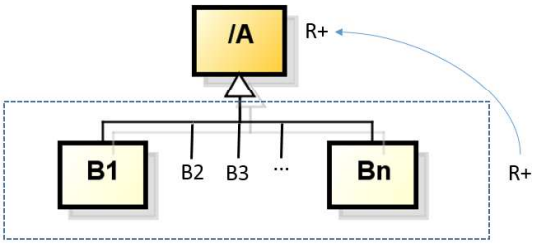

Figura 4.2 – Caso 1 de Rigidez em Tipos Derivados por União

Prova 1: Dado que A é derivado por união, todas as instância de A são instâncias de um subtipo B_n de A. Em outras palavras, para todo x, m tal que $x \in ext(A,m)$ existe um B_n tal que $x \in ext(B_n,m)$. Suponha que existe um m' tal que $x \notin ext(A,m')$. Então $x \notin ext(B_n,m')$ (caso contrário, uma vez que B_n é um subtipo de A, se $x \in ext(B_n,m')$. Como B_n é rígido, temos que então $x \in ext(B_n, m')$ e, portanto, $x \in ext(A,m')$).

Caso 2: Alguns Tipos Rígidos e Outros Não-Rígidos – ($DS \cap C_{R+} \neq \varnothing$, $DS \cap C_{R-} \neq \varnothing$)

Supondo que ambos $SC_R \neq \varnothing$ e $SC_{AR} \neq \varnothing$ (i.e., existe pelo menos um subtipo rígido e um subtipo anti-rígido de A no DS). Nesse caso A é ou Rígido ou Semi-Rígido como demonstrado na **Figura 4.3**.

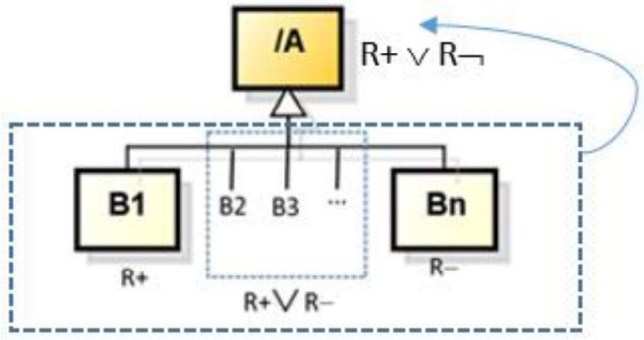

Figura 4.3 – Caso 2 de Rigidez em Tipos Derivados por União

Prova 2: Dado que existe um B_{iR} que é rígido e que B_{iR} especializa A, então todas as instâncias de B_{iR} são necessariamente instâncias de /A. Logo, A não pode ser anti-rígido. Agora, para as instâncias dos tipos de $SC_{AR,}$ pode-se ter dois caso: (i) o caso no qual cada instância x de qualquer tipo $B_{iAR} \in SC_{AR}$ deixa de ser instância de B_{iAR} necessariamente se torna uma instância de um outro tipo $B_{jAR} \in SC_{AR.}$ Nesse caso, x nunca deixa de ser instância de A, ele meramente alterna entre os tipos de SC_{AR}. Nesse caso, A é rígido. (ii) no caso em que existe pelo menos uma instância de $B_{iAR} \in SC_{AR}$ que deixa de ser instância de B_{iAR} sem se tornar instância de um outro tipo $B_{jAR} \in SC_{AR.}$ Nesse caso, A é não-rígido. Uma vez que ele não pode ser anti-rígido, então ele é necessariamente semi-rígido.

Caso 3: Todos Tipos Não-Rígidos ($DS \subseteq C_{R-}$)

Supondo-se que $SC_R = \varnothing$, i.e., todos subtipos de A são não-rígidos então A é rígido ou não-rígido, i.e., não se pode concluir nada sobre a rigidez de A.

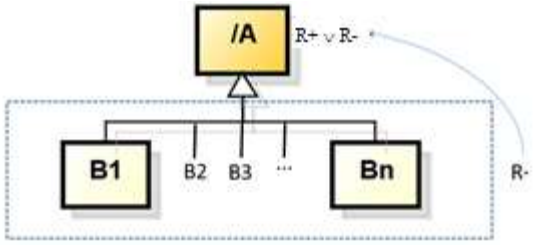

Figura 4.4 – Caso 3 de Rigidez em Tipos Derivados por União

Prova 3: Uma vez mais tem-se os dois casos anteriormente mencionados de acordo com SC_{AR}: (i) o caso no qual cada instância x de qualquer tipo $B_{iAR} \in SC_{AR}$ deixa de ser instância de B_{iAR} necessariamente se torna uma instância de um outro tipo $B_{jAR} \in SC_{AR.}$ Nesse caso, x nunca deixa de ser instância de A, ele meramente alterna entre os tipos de SC_{AR}. Nesse caso, A é rígido. (ii) no caso em que existe pelo menos uma instância de $B_{iAR} \in SC_{AR}$ que deixa de ser instância de B_{iAR} sem se tornar instância de um outro tipo $B_{jAR} \in SC_{AR.}$ Nesse caso, A é não-rígido.

4.4.2 Geração dos Padrões de Tipos Derivados por União com base nas Combinações Válidas de Estereótipos em OntoUML

De acordo com o processo estabelecido, na etapa 3 aplica-se para cada caso de rigidez as variações de estereótipos possíveis. Os estereótipos possíveis para o tipo derivado são inferidos considerando os aspectos rigidez, princípio de identidade e dependência.

A **Tabela 4.1** apresenta as variações possíveis para o caso 1 em que todos os tipos da DS são rígidos e o tipo derivado também é rígido.

Tabela 4.1 – Padrões de Tipos Derivados por União de OntoUML para o Caso 1

Padrão	Ocorrência dos Estereótipos no DS	Estereótipo do Tipo Derivado
1	$DS \subseteq (C_{Kind} \cup C_{Collective} \cup C_{Quantity})$	Category
2	$DS \subseteq SC_R$, $DS \cap C_{Category} \neq \varnothing$	Category
3	$DS \subseteq C_{Subkind}$	Subkind, Category

A **Tabela 2.1** apresenta as generalizações e especializações válidas para cada estereótipo de OntoUML. Nela temos que **Substance Sortals** (i.e. **Kinds, Collectives** e **Quantities**) e **Category** só podem ter como supertipo um **Category** ou um **Mixin**. Como para o caso 1, o tipo de objeto derivado deve ser rígido, então o tipo derivado deve ser um **Category**, como pode-se perceber nos **padrões 1 e 2** da **Tabela 4.1**.

O **padrão 3** assume diferentes categorizações para o tipo derivado de acordo com duas possíveis variações. Se todos os **Subkinds** do DS são subtipos de um mesmo **Substance Sortal**, então A será outro **SubKind** (uma vez que **Substance Sortals** não podem ser derivados como descrito na Restrição 2). Caso contrário, se dois dos **Subkinds** do DS tem como supertipo **Substance Sortals** distintos, então o tipo de objeto derivado será um **Category**.

A **Tabela 4.2** apresenta as variações possíveis do caso 2 em que se tem um DS com pelo menos um tipo rígido e pelo menos um tipo não-rígido.

Tabela 4.2 – Padrões de Tipos Derivados por União de OntoUML para o Caso 2

Padrão	Ocorrência dos Estereótipos no DS	Estereótipo do Tipo Derivado
4	$DS \cap C_{R+} \neq \emptyset$, $DS \cap C_{R-} \neq \emptyset$, $DS \cap (C_{Category} \cup C_{Mixin} \cup C_{RoleMixin}) \neq \emptyset$	Category, Mixin
5	$DS \cap C_{R+} \neq \emptyset$, $DS \cap C_{R-} \neq \emptyset$, $DS \cap (C_{Kind} \cup C_{Collective} \cup C_{Quantity}) \neq \emptyset$	Category, Mixin
6	$DS \subseteq (C_{Role} \cup C_{Subkind})$, $DS \cap C_{Role} \neq \emptyset$, $DS \cap C_{Subkind} \neq \emptyset$	Subkind, Mixin
7	$DS \subseteq (C_{Phase} \cup C_{Subkind})$, $DS \cap C_{Phase} \neq \emptyset$, $DS \cap C_{Subkind} \neq \emptyset$	Subkind, Mixin

Como já discutido no Capítulo 2, um tipo Rígido não pode ter um supertipo Anti-Rígido. Além disso, supertipos de **Non-Sortals** e dos **Substance Sortals** podem somente ser **Non-Sortals**. Sendo assim, tanto os **padrões 4** (que cobre a ocorrência de pelo menos um tipo **Non-Sortal** para o caso 2) como para o **padrão 5** (que cobre a ocorrência de pelo menos um **Substance Sortal** para o caso 2) o tipo derivado deve ser um **Non-Sortal** Rígido (**Category**) ou Semi-Rígido (**Mixin**). A variação quanto a categorização do tipo derivado nos dois padrões se dá de acordo com o comportamento das instâncias dos subtipos Anti-Rígidos de A. Se as instâncias dos subtipos Anti-Rígidos de A migrarem somente entre si (i.e., nunca migrarem para fora do DS) então A será rígido e por consequência será um **Category**. Caso contrário, (i.e., se as instâncias de um subtipo Anti-Rigido de A puderem migrar para um outro tipo que não é subtipo de A), então A será um **Mixin**. A **Figura 4.5** demonstra exemplos de estruturas válidas para cada um dos estereótipos possíveis do padrão 4.

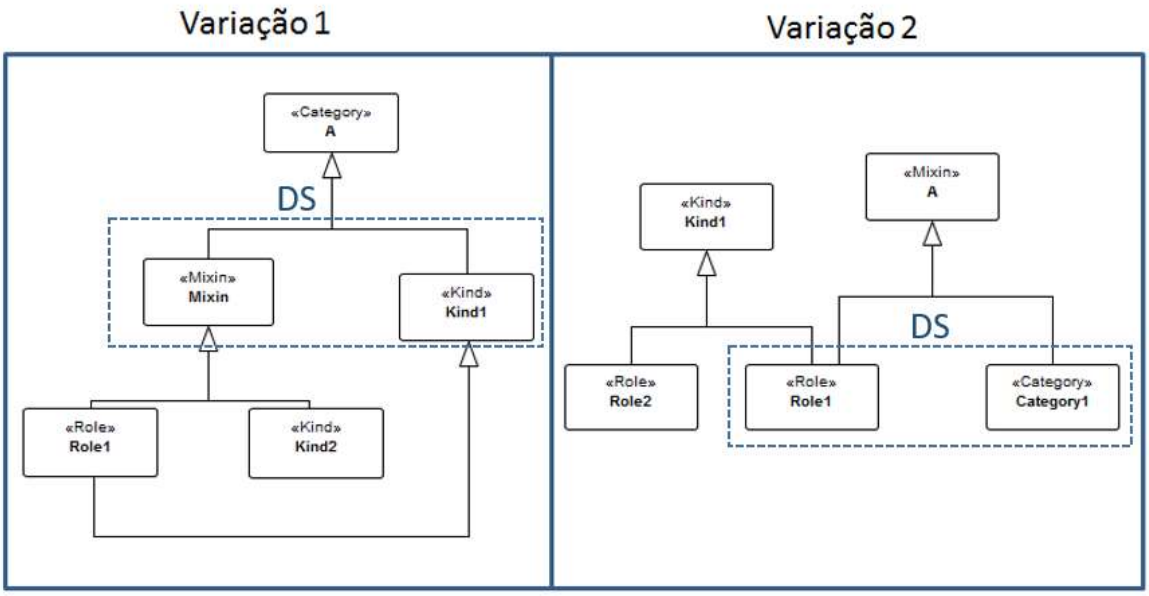

Figura 4.5- Variações do Padrão 4

Como pode-se perceber na variação 1, caso as instâncias do Role1 nunca deixarem de ser instâncias de A (isso ocorre porque quando deixam de ser instâncias do Role1 são instâncias do Kind1 que também é subtipo de A), o tipo de objeto derivado A será um **Category**. Já na variação 2, as instâncias do Role1 podem migrar para o Role2 que não é subtipo de A, fazendo que o tipo derivado seja um **Mixin**.

Para os padrões 6 e 7, segundo a Tabela 2.1, os supertipos válidos comuns a **Subkind**, **Role** e **Subkind** e **Phase** são **Substance Sortals**, **Subkind** e **Mixin**. **Substance Sortals** são desconsiderados como possíveis tipos derivados pela Restrição 2. Sendo assim, restam as categorizações **Mixin** e **Subkind** como opções válidas para o padrão 6 e 7. O tipo derivado será um **Subkind** se as instâncias dos tipos Anti-Rígidos nunca deixarem de ser instâncias de A (dessa forma A se mantém Rígido). Caso contrário o tipo derivado será um **Mixin**.

Para clarear o entendimento desse padrão, a **Figura 4.6** apresenta o comportamento modal das instâncias em cada caso possível. Quando as instâncias dos subtipos anti-rígidos de A migram somente entre as partes anti-rígidas do DS, então tem-se um **Category**. Se algumas das instâncias de alguma das partes anti-rígidas do DS tem possibilidade de migrar para fora do DS (deixando de ser instância do DS) então o tipo derivado é um **Mixin**.

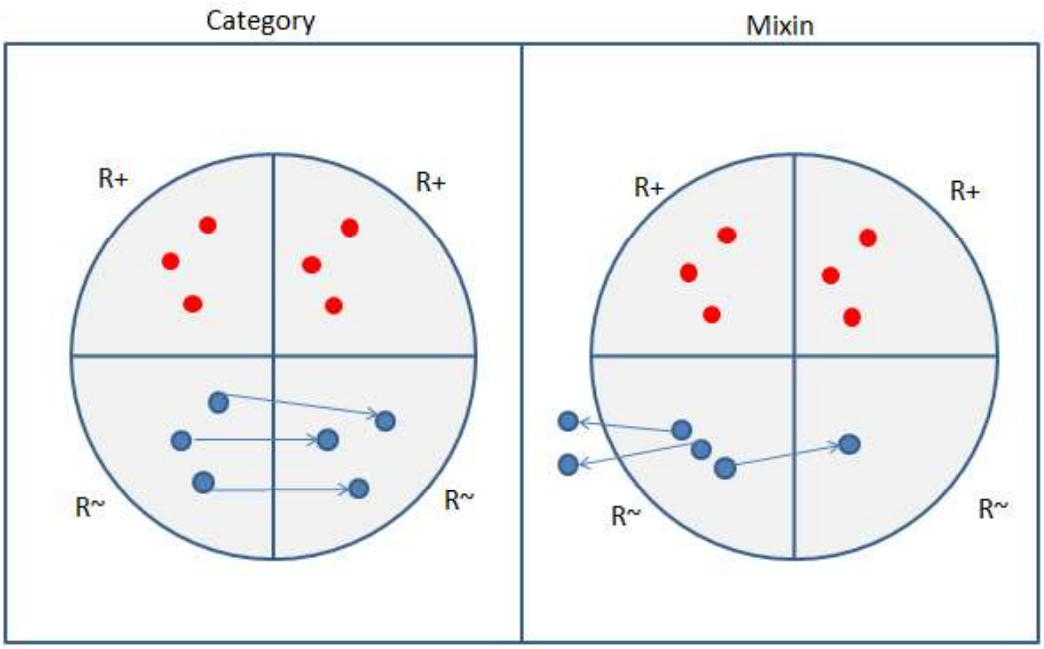

Figura 4.6 - Casos Possíveis de União de Mixins

A **Tabela 4.3** apresenta os padrões para o caso 3 em que o DS contém somente tipos Não-Rígidos.

Tabela 4.3 - Padrões de Tipos Derivados por União de OntoUML para o Caso 3

Padrão	Ocorrência dos Estereótipos no DS	Estereótipo do Tipo Derivado
8	$DS \subseteq C_{R-}$, $DS \cap C_{Mixin} \neq \emptyset$	Category, Mixin
9	$DS \subseteq C_{R-}$, $DS \cap C_{RoleMixin} \neq \emptyset$, $DS \cap C_{Mixin} = \emptyset$	RoleMixin, Mixin, Category
10	$DS \cap C_{Role} \neq \emptyset$, $DS \cap C_{Phase} \neq \emptyset$, $DS \subseteq (C_{Role} \cup C_{Phase})$	Phase, Role
11	$DS \subseteq C_{Role}$	Role, RoleMixin
12	$DS \subseteq C_{Phase}$	Phase

O **padrão 8** cobre o caso em que se tem pelo menos um Mixin no DS. Mais uma vez o resultado é determinado pelo comportamento das instâncias dos subtipos Anti-Rígidos de A. Se as instâncias dos subtipos Anti-Rígidos de A nunca migrarem para fora de A, então A será um **Category**. Caso contrário ele será um **Mixin** (veja Figura 4.6).

O **padrão 9** tem pelo menos um **Role Mixin** e nenhum **Mixin** como membro do DS. Nesse caso, temos três possibilidades. Na primeira, se todas as instâncias de A puderem em algum

mundo possível deixar de ser de A, então A será um **Role Mixin**. Se algumas delas puderem deixar de ser de A e outras não, A será um **Mixin**. Se nenhuma das instâncias de A nunca migrarem para for a de A, A será um **Category**.

O **padrão 10** compreende um DS com **Phase(s)** e **Role(s).** Para determinar o resultado precisa-se considerar o princípio de identidade e a Dependência dos tipos. Primeiramente se pelo menos dois dos membros do DS não são subtipos de um mesmo **Substance Sortal**, então o tipo derivado seria um **Non-Sortal** anti-rígido. O único **Non-Sortal** anti-rígido definido em (Guizzardi 2005) é o **Role Mixin**. O fato é que nesse caso temos **Roles** e **Phases** compondo o DS, o que difere da definição de Guizzardi (2005) na qual um **Role Mixin** deve agregar propriedades somente de **Roles**. Como pode-se perceber, não há uma categorização OntoUML para um **Non-Sortal** anti-rígido que agrega propriedades de **Roles** e de **Phases**. Sendo assim, nesse trabalho considera-se inválida a união de **Phases** e **Roles** que sejam subtipos de **Substance Sortals** distintos. Caso contrário (se todos os tipos do DS são subtipos do mesmo **Substance Sortal**), considera-se a meta-propriedade de dependência.

Todo **Role** é externamente dependente (D+) por definição. **Phases** por sua vez podem ser tanto dependentes quanto independentes (D-). Um **Phase** dependente é um **Phase** que herdou sua dependência de um **Role**. Se aplicarmos uma derivação por união entre **Roles** e **Phases** marcados com D+, então teremos como tipo derivado um **Sortal** Anti-Rígido marcado com D+, uma vez que todas as suas instâncias serão externamente dependentes. Não há, nesse caso, argumentos que permitam definir qual a categorização para o tipo derivado com base nas meta-propriedades dos tipos. Sendo assim, considera-se válido para esse caso que o tipo derivado seja tanto um **Role** como um **Phase**. Já, no caso em que pelo menos um dos **Phases** que compõem o DS seja marcado com D- então a independência desse tipo será propagada para o supertipo e então o tipo derivado será um **Phase**.

No **padrão 11**, o tipo derivado será um **Role** quando os membros do DS forem subtipos de um mesmo **Substance Sortal**. Caso contrário o tipo derivado será um **Role Mixin**. Analogamente, para o **padrão 12** o tipo derivado será um **Phase** quando os membros do DS forem subtipos de um mesmo **Substance Sortal**. Caso contrário, mais uma vez tem-se um caso de **Non-Sortal** Antí-Rígido não definido na linguagem. Em Fonseca (2015) foi proposto o conceito de **Phase Mixin** que é um **Non-Sortal** anti-rígido que agrupa propriedades de **Phases**. Nesse trabalho

não se considera tal opção por que se utiliza tanto para definição dos padrões como para implementação a especificação de OntoUML definida em Guizzardi (2005).

4.5 Padrões de Tipos Derivados por Exclusão em OntoUML

4.5.1 Valores de Rigidez em Padrões de Tipos Derivados por Exclusão

A derivação por exclusão segue a estrutura apresentada na **Figura 4.7** em que um tipo derivado por exclusão e o tipo base fazem parte do mesmo *Generalization Set* marcado como *disjoint* e *complete*.

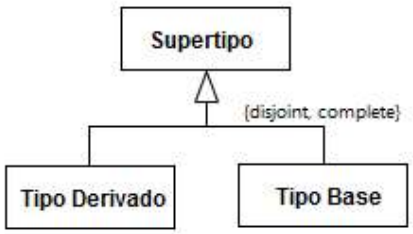

Figura 4.7 – Estrutura de Tipo Derivado por Exclusão

Em derivações por exclusão, algumas configurações de valores quanto a rigidez são inconsistentes. O caso 2 e o caso 9 da **Tabela 4.4** são logicamente inconsistentes porque tanto um tipo rígido como um tipo semi-rígido não podem ser subtipo de um tipo anti-rígido. A seguir prova-se os casos consistentes apresentados na **Tabela 4.4**.

Tabela 4.4 – Combinações de Rigidez para tipo Derivado por Exclusão

Caso	Tipo Base	**Supertipo**	**Tipo Derivado**
1	**R+**	R+	R+
~~**2**~~	~~**R+**~~	~~R~~	~~Logicamente Inconsistente~~
3	**R+**	R¬	R–
4	**R~**	R+	R–
5	**R~**	R~	R~
6	**R~**	R¬	R+, R¬
7	**R¬**	R+	R–
~~**8**~~	~~**R¬**~~	~~R~~	~~Logicamente Inconsistente~~
9	**R¬**	R¬	R+,R–

Considere A o Supertipo, B o tipo base e C o tipo Derivado, então:

Caso 1: No caso 1, tem-se A e B como rígidos. A seguir, demonstra-se que C deve ser rígido também.

Prova 4: Se o A e o B são Rígidos sua extensão é invariante, i.e., para todo m,m' $\in$ M, ext(A,m)= ext(A,m') e ext(B,m) = ext(B,m'). Uma vez que ext(A,m) = ext(B,m) $\cup$ ext(C,m), para todo m $\in$ M, deve-se concluir que a extensão de C é também um mundo invariante, i.e., ext(C,m) = ext(C,m'), para todo m,m'. Alternativamente, pode-se assumir que C é não-rígido. Se esse fosse o caso existiriam x,m,m' tal que x $\in$ ext(C,m) e x $\notin$ (C,m'). Mas uma vez que C é um subtipo de A então sabe-se que x $\in$ ext(A,m) e x $\in$ ext(A,m'). Uma vez que o *generalization set* é completo, tem-se que x $\in$ ext(B,m'). Uma vez que B é rígido então x $\in$ ext(B,m). Tendo que x é uma instância tanto de B e C em m, e que B e C são disjoint, tem-se uma contradição, então, conclui-se que C não pode ser não-rígido, i.e., ele deve ser rígido.

Caso 3: No caso 3 tem-se o Supertipo semi-rígido e o tipo base Rígido. A seguir, demonstra-se que o tipo derivado deve ser Não-Rígido.

Prova 5: Supondo-se C rígido para o caso 3, então tem-se um caso, que é análogo ao caso 1. Então novamente tem-se mais uma vez que se B e C são rígidos o mesmo serve para A, o que

é uma contradição (nossa primeira premissa é que A é Semi-Rígido). Então, C deve ser não-rígido.

Case 4,5,6 : Supondo-se que B é Anti-Rígido. Mais uma vez, o valor de rigidez de C pode ser derivado de A. Se A é rígido então C deve ser não-rígido; Se A é também Anti-Rígido então o mesmo serve para C; se A é Semi-Rígido então C deve ser ou rígido ou Semi-Rígido.

Prova 6: Uma vez que B é Anti-Rígido então existe um x bem como m,m' $\in$ M tal que x $\in$ ext(B,m) e x $\notin$ ext(B,m'). Uma vez que B e C são disjoint, sabe-se que x $\notin$ ext(C,m), e uma vez que B é um subtipo de A sabe-se que x $\in$ ext(A,m). Uma vez que A é Rígido então sua extensão é invariante e então x $\in$ ext(A,m'). Uma vez que esse *generalization set* é completo, tem-se ext(A,m')=ext(B,m') $\cup$ ext(C,m'). Dado que x $\notin$ ext(B,m') deve-se ter x $\in$ ext(C,m'). Isso demonstra que a extensão de C não é invariante e, então, que ele não pode ser um tipo Rígido. Logo, C é Não-Rígido.

Prova 7: Se A é Anti-Rígido então C não pode ser Rígido uma vez que um tipo Anti-Rígido não pode ser supertipo de um Rígido. Além disso, C não pode ser um tipo Semi-Rígido porque (por definição) C teria que ter como subtipo um tipo Rígido D. Nesse caso D seria um (indireto) subtipo de A e então ter-se-ia um tipo Anti-Rígido como supertipo de um Rígido. Logo, deve-se concluir que C deve ser Anti-Rígido nesse caso.

Prova 8: Se A é Semi-Rígido então (por definição) A deve ter como subtipo um tipo rígido. Uma vez que o *Generalization Set* envolvendo B e C (e tendo A como supertipo comum) é completo então ou B ou um de seus subtipos é rígido ou C ou um dos seus subtipos é rígido. B é Anti-rígido e seus subtipos não podem ser rígidos. Então, ou C é Rígido ou um dos seus subtipos é rígido. Se C é não-rígido então a única maneira de um de seus subtipos poder ser rígido é se C for Semi-rígido. Logo conclui-se que C é ou Rígido ou Semi-Rígido.

Case 4,5 e 6 : Supondo-se que B é Anti-Rígido. Mais uma vez, o valor de rigidez de C pode ser derivado de A. Se A é rígido então C deve ser não-rígido; Se A é também Anti-Rígido então o mesmo serve para C; se A é Semi-Rígido então C deve ser ou rígido ou Semi-Rígido.

Case 7 e 9 : Finalmente tendo B como Semi-Rígido então: se A é rígido C é não-rígido e se A é Semi-Rígido então C pode assumir qualquer valor.

Prova 9: Uma vez que B é Semi-Rígido então existe um x bem como m,m' $\in$ M tal que x $\in$ ext(B,m) e x $\notin$ ext(B,m'). Uma vez que B e C são *disjoint* sabe-se que x $\notin$ ext(C,m), e uma vez que B é um subtipo de A sabe-se que x $\in$ ext(A,m). Uma vez que A é Rígido, sua extensão é invariante e então tem-se que x $\in$ ext(A,m'). Considerando que o *generalization set* é completo, tem-se que ext(A,m') = ext(B,m') $\cup$ ext (C,m'). Dado que x $\notin$ ext(B,m') deve-se ter ext(C,m'). Isso demonstra que a extensão de C é não invariante e, logo, que ele não pode ser Rígido. C então só pode ser Não-Rígido.

Prova 10: Uma vez que A é semi-rígido então existem instâncias de A que são invariantes (i.e., para todo x e m tal que que m $\in$ M e x $\in$ ext(A,m) então x $\in$ ext(A,M) e outras não (i.e., existe x', x'' bem como um m,m' tal que x' $\in$ ext(A,m) e x'' $\notin$ ext(A,m')). Tendo que um dado x $\in$ ext(C,m) e C é subtipo de A então x $\in$ ext(A,m). Logo se C for rígido, então x será invariante em relação a C e em relação a A. Uma vez que existem instâncias de A que são invariantes, então C se mantém consistente sendo Rígido. Se C for Anti-Rígido ou Semi-Rígido então tem-se dois casos possíveis. No primeiro tem-se: para todo x tal que se x $\in$ ext(C,m) e x $\notin$ ext(C,m') então x $\in$ ext(B,m'). Nesse caso como B e C são subtipos de A, x será invariante em relação a A, o que é consistente porque A tem instâncias invariantes. No segundo caso existe um x tal que x $\in$ ext(C,m) e x $\notin$ ext(C,m') e x $\notin$ ext(B,m'). Nesse caso, como o generalization set é complete então existe um x' bem como um m,m' $\in$ ext(A,m) e x $\notin$ ext(A,m'). Como A tem sua parte não invariante então C pode ser também Semi-Rígido e Anti-Rígido. Logo C pode ser Rígido, Semi-Rígido ou Anti-Rígido.

4.6 Geração dos Padrões de Tipos Derivados por Exclusão com base nas Combinações Válidas de Estereótipos em OntoUML.

A **Tabela 4.5** apresenta os padrões válidos para o caso 1 da **Tabela 4.4**. Como pode-se perceber, ela define as combinações de estereótipos OntoUML possíveis para Supertipo (A) e um Tipo Base (B), determinando então o estereótipo do Tipo Derivado (C).

Tabela 4.5 – Padrões de Tipos Derivados por Exclusão de OntoUML para o Caso 1

Padrão	A	B	C
13	Kind, Collective, Quantity, Subkind	Subkind	Subkind
14	Category	Kind, Subkind, Category	Subkind, Category

No padrão 13 tem-se A (o supertipo de B e C) categorizado ou como um **Substance Sortal** ou como um **Subkind**. Sabe-se que C deve rígido, logo C deve ser ou um **Substance Sortal**, **Subkind** ou **Category**. O tipo derivado C não poderia ser um **Substance Sortal** porque C é um subtipo de A e em OntoUML um **Substance Sortal** não pode ser subtipo de outro **Substance Sortal.** C também não pode ser um **Category** porque **Non-Sortals** não podem ser subtipos de **Sortals**. Logo C deve ser um **Subkind**.

Nos **Padrão 14** elimina-se somente os **Substance Sortals** como opção para a categorização do tipo derivado porque esses não podem ser derivados. Tem-se então, **Subkind** e **Category** como opção.

O caso 3 tem um tipo Semi-Rígido como supertipo, ou seja, um **Mixin**. Nesse caso, independente do estereótipo de B, qualquer categorização não-rígida para C torna consistente o padrão. Logo tem-se um único padrão para o caso 3 como segue:

Padrão 15: A(Mixin) ∧ B(Não-Rígido) então C(Mixin) ∨ C(RoleMixin) ∨ C(Role) ∨ C(Phase)

A **Tabela 4.6** apresenta os padrões válidos para o caso 4:

Tabela 4.6 - Padrões de Tipos Derivados por Exclusão de OntoUML para o Caso 4

Padrão	A	B	C
16	Category	RoleMixin, Role, Phase	RoleMixin, Role, Phase, Mixin
17	Kind	Role	Role
18	Kind	Phase	Phase
19	Subkind	Role	Role
20	Subkind	Phase	Phase

No **padrão 16** qualquer categorização de tipo não rígido se aplica (i.e não há restrições a serem feitas). Do padrão 1 até o 19 tem-se sempre *generalization sets* homogêneos (i.e. formados por tipos com a mesma categorização). Isso ocorre porque um subtipo de um **Sortal** só pode ser um outro **Sortal** e porque **Phases** ocorrem somente em **Phase Partitions**. Ou seja, um **Phase** e um **Role** não podem fazer parte de um mesmo *generalization set*.

A **Tabela 4.7** apresenta os padrões válidos para o caso 5 da **Tabela 4.4**.

Tabela 4.7 - Padrões de Tipos Derivados por Exclusão de OntoUML para o Caso 5

Padrão	A	B	C
20	RoleMixin	Role, RoleMixin	Role, RoleMixin
21	Role	Role	Role
22	Role	Phase	Phase
23	Phase	Role	Role
24	Phase	Phase	Phase

No **padrão 20**, tanto o tipo base comRolo o derivado são ou **Role Mixin** ou **Role** porque o supertipo é um **Role Mixin** e Phases não podem ser subtipo de **Role Mixin**. Do **padrão 21** até o **padrão 24**, novamente tem-se a mesma restrição aplicada no caso 4 em que **Phases** e **Roles** não podem compor o mesmo *generalization set.*

O caso 7 tem um tipo Semi-Rígido comotipo base, ou seja, um **Mixin**. Nesse caso, qualquer categorização não-rígida para C torna consistente o padrão. Logo tem-se um único padrão para o caso 7 como segue

Padrão 25: A(Category) ∧ B(Mixin) então C(Mixin) ∨ C(RoleMixin) ∨ C(Role) ∨ C(Phase)

O único tipo Semi-Rígido de OntoUML é o **Mixin**. Se um tipo Rígido for supertipo de um **Mixin** esse tipo deve ser um **Category (**uma vez que um **Non-Sortal** só podem ter como supertipo outro **Non-Sortal)**.

Para o caso 9, tem-se também um único padrão possível porque esse caso tem um **Mixin** como supertipo e como tipo base. Qualquer categorização (com exceção dos **Substance Sortals**) é válida nesse caso.

Padrão 26: A(Mixin) ∧ B(Mixin) então C(Category) ∨ C(Mixin) ∨ C(RoleMixin) ∨ C(Subkind) ∨ C(Role) ∨ C(Phase)

4.7 Padrões de Tipos Derivados por Interseção em OntoUML

Tipos derivados por interseção podem ser aplicados em qualquer combinação de valores quanto a Rigidez. Sendo assim, separa-se três casos possíveis que compreendem todas as combinações possíveis em um DS. Para todos os casos supõe-se um tipo B derivado por interseção de um DS=$\{A_1,...,A_n\}$.

Caso 1 : Todos Tipos do DS são Rígidos - ($DS \subseteq C_{R+}$)

Se todos os tipos do DS=$\{A_1,...,A_n\}$ forem Rígidos então a Rigidez é propagada para o tipo B derivado por interseção.

Prova 11: Para todo $x \in ext(B,m)$ tem-se que $x \in ext(A_1,m) \cap,\ldots, \cap ext(A_n,m)$. Em outras palavras, para todo $A_i \in DS$, $x \in ext(A_i,m)$. Supondo-se que que B é Não-Rígido então existe uma instância x' de B em um mundo m' tal que $x' \notin ext(B,m')$. Se esse fosse o caso então $x' \notin ext(A_1,m') \cap,\ldots, \cap ext(A_n,m')$ o que é absurdo porque todos os membros do DS=$\{A_1,...,A_n\}$ são Rígidos. Logo B deve ser Rígido.

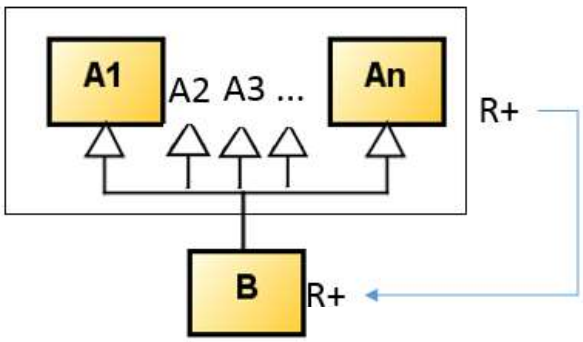

Figura 4.8 – Caso 1 de Rigidez em Tipos Derivados por Interseção

Caso 2: Ocorrência de um tipo Anti-Rígido – ($DS \cap C_{R^-} \neq \emptyset$)

Se apenas um elemento do DS for Anti-Rígido então B será Anti-Rígido também.

Prova 12: Temos que qualquer que seja $m \in M$ então $ext(B,m) = ext(A_1,m) \cap,\ldots, \cap ext(A_n,m)$. Se existe um A_i que é anti-rígido, então para todo x tal que $x \in ext(A_i,m)$, existe um m' tal que $x \notin ext(A_i,m')$. Perceba que nesse m', $x \notin ext(A_1,m') \cap,\ldots, \cap ext(A_n,m')$ (visto que qualquer elemento que pertence a interseção, por definição, pertence a cada um dos conjuntos que formam a interseção). Como, também por definição, todos os elementos que formam a interseção pertencem a A_i, temos que todos esses elementos da interseção tem esse comportamento que acabamos de descrever comum a todas as instancias de Ai, i.e., todos eles necessariamente deixarão de pertencer a interseção. Como consequência, a interseção (e, portanto, B) define um tipo anti-rígido. .

Caso 3: Ocorrência de Rígidos e Semi-Rígidos – ($DS \cap C_{R^\neg} \neq \emptyset$, $DS \subseteq (C_{R^\neg} \cup C_{R^+})$)
Quando todos os tipos no DS são rígidos ou semi-rígidos sendo que pelo menos um deles é semi-rígido então o tipo derivado poderá assumir qualquer valor quanto a rigidez.

Prova 13: Uma vez que $DS \subseteq (C_{R^\neg} \cup C_{R^+})$ então para, todo $A_i \in DS$, existe pelo pelos um x tal que para qualquer $m \in M$ e $x \in ext(A_i,m)$. Portanto é possível que $B = A_1 \cap ... \cap A_n$ classifique somente aquelas instâncias de A_i que necessariamente o são. Em outras palavras, temos que em todos os mundos possíveis, os indivíduos que fazem parte da interseção de todos os tipos $A_i \in DS$ instanciam necessariamente todos esses A_is. Por outro lado, como $DS \cap C_{R^\neg}$

$\neq \varnothing$, isso significa que existe pelo menos um $A_i \in$ DS que é semi-rígido, ou seja, que possui pelo menos um subconjunto dos seus elementos que necessariamente deixam de ser A_i, Suponha que $B = A_1 \cap ... \cap A_n$ seja formado exatamente por elementos nesse subconjunto anti-rígido de A_i. Nesse caso, como discutido na prova do caso anterior, todos os elementos de B necessariamente deixarão de ser instancias de B (i.e., B é anti-rígido). Novamente, como DS $\cap C_{R\neg} \neq \varnothing$, i.e., pelo menos um $A_i \in$ DS que é semi-rígido, A_i possui elementos que instanciam necessariamente A_i e outros que necessariamente deixarão de instanciar A_i. Portanto, é possível que exatamente uma combinação desses elementos venha a formar a extensão de B em um mundo possível. Nesse caso, B será também semi-rígido. Em resumo, B poderá nesse caso assumir qualquer valor quanto a rigidez.

4.7.1 Geração dos Padrões de Tipos Derivados por Interseção com base nas Combinações Válidas de Estereótipos em OntoUML

A primeira restrição a ser imposta para qualquer padrão de derivação por interseção é que um **Substance Sortal** não pode aparecer como membro de um DS. Pode-se chegar a essa conclusão analisando as três configurações de um DS contendo **Substance Sortals**:

No primeira delas teríamos um DS formado por mais de um **Substance Sortal**. Nesse caso não podemos ter um tipo derivado por interseção válido porque suas instâncias teriam mais de um princípio de identidade.

Na segunda configuração teríamos um **Substance Sortal** A compondo o DS junto com um **Non-Sortal** *B*. Nesse caso, se tivermos um tipo derivado por interseção C, então ou A seria subtipo de B (o que é excluído pela Restrição Geral 1) ou então B será um supertipo de outros **Substance Sortals**. Se esse fosse o caso então mais uma vez as instâncias de C teriam mais de um princípio de identidade.

Na última configuração possível tem-se um **Substance Sortal** A em um DS com Sortals não provedores de princípio de identidade. Os Sortals do DS não provedores de identidade ou serão subtipos de A (o que é restringido pela Restrição Geral 1) ou então serão subtipo de um outro **Substance Sortal**. Se esse fosse o caso então mais uma vez as instâncias do tipo derivado por

interseção teriam mais de um princípio de identidade. A **Tabela 4.8** apresenta os padrões para o caso 1.

Tabela 4.8 - Padrões de Tipos Derivados por Interseção de OntoUML para o Caso 1

Padrão	Ocorrência dos Estereótipos no DS	Estereótipo do Tipo Derivado
27	$DS \subseteq C_{Category}$	Category, Subkind
28	$DS \subseteq C_{R+}$, $DS \cap C_{Subkind} \neq \emptyset$	Subkind

Uma vez que se determinou que **Substance Sortals** não podem ser derivados, então restam **Category** e **Subkind** como opções de categorização para um tipo de objeto derivado por interseção em um DS formado somente por tipos rígidos. O tipo derivado por interseção em um DS formado somente por **Categories** poderá ser um **Category** ou um **Subkind** (**Padrão 27**).

Já em DS que contenham **Subkinds** o tipo derivado terá que ser outro **Subkind**. Isso porque um **Subkind** só podem ter como subtipo rígido outro **Subkind**. Sendo assim, qualquer ocorrência de um **Subkind** em um DS formado por tipos rígidos leva o tipo derivado por interseção a ser categorizado como **Subkind** (**Padrão 28**).

A **Tabela 4.9** apresenta os padrões para o caso 2.

Tabela 4.9 - Padrões de Tipos Derivados por Interseção de OntoUML para o Caso 2

Padrão	Ocorrência dos Estereótipos no DS	Estereótipo do Tipo Derivado
29	$DS \subseteq C_{Role}$	Role
30	$DS \subseteq C_{Phase}$	Phase
31	$DS \cap C_{RoleMixin} \neq \emptyset$, $DS \subseteq (C_{Non\text{-}Sortal})$	RoleMixin
32	$DS \cap C_{R\sim} \neq \emptyset$, $(DS \cap C_{Sortal} \neq \emptyset)$*	Role, Phase

O **padrão 29** contempla um DS formado somente por **Roles**. O tipo derivado por interseção nesse caso herda a dependência dos **Roles** que compõem o DS. Por essa razão o tipo derivado deve ser outro **Role**.

Em uma interseção de um DS formado somente por **Phases** (**padrão 30**), os membros do DS devem fazer parte de *generalization sets* distintos (uma vez que os membros de um *Phase Partition* devem ser disjuntos). O tipo derivado nesse caso será um tipo derivado definido com

base nas propriedades intrínsecas dos tipos que formam o DS. Logo, defende-se que esse tipo deve ser um outro **Phase**. Embora esse padrão infrinja a regra que **Phases** devam ocorrem sempre em *Phase Partitions,* nesse caso, o tipo derivado representa a reificação das propriedades que caracterizam cada um dos **Phases** do DS. Por essa razão sua categorização como **Phase** se justifica.

Se houver a ocorrência de um tipo A categorizado como **Role Mixin** em um DS formado somente por **Non-Sortals** então o tipo derivado por interseção será outro **Role Mixin** (**padrão 31**). Isso ocorre porque um tipo Anti-Rígido só pode ter outro tipo Antí-Rígido como subtipo. E como todas as instâncias de A serão também instância do tipo derivado C, e A agrupa tipos provedores de princípio de Identidade distintos, então C também estará agrupando tipos com princípio de Identidade distintos. Logo C será um **Role Mixin**.

Para contemplar todos os casos possíveis de interseção em um DS com membros não-rígidos restam diversos casos não contemplados nos padrões anteriores. Os casos não contemplados anteriormente são agrupados no **padrão 32**, que tem pelo menos um **Sortal** no DS. Além disso, o DS, nesses casos, é não homogêneo em relação a categorização dos tipos (ao contrário dos padrões 29 e 30). Ou seja, caso haja pelo menos um **Sortal** no DS e o DS não satisfaça o padrão 29 e nem o padrão 30 então o padrão 31 será aplicado. Como no **padrão 32** tem-se a ocorrência de um **Sortal** e de um tipo anti-rígido, o tipo derivado deverá ser um **Sortal** (uma vez que **Sortals** só podem ser subtipos de **Sortals**) e Anti-Rígido (i.e., **Phase** ou **Role**). Em nenhuma das possíveis configurações para o **padrão 32** tem-se argumentos baseados em meta-propriedades que levem a categorização do tipo como uma das duas categorizações específicas (i.e., **Phase** ou **Role)**. Logo, são consideradas as duas possibilidades como válidas.

A apresenta os padrões para o caso 3.

Tabela 4.10 – Padrões de Tipos Derivados por Interseção de OntoUML para o Caso 3

Padrão	Ocorrência dos Estereótipos no DS	Estereótipo do Tipo Derivado
33	$DS \subseteq (C_{Mixin} \cup C_{Category})$	Role, Phase, Subkind, Category, RoleMixin,Mixin
34	$DS \cap C_{R\neg} \neq \emptyset$, $DS \cap (C_{Subkind} \cup C_{Kind} \cup C_{Quantity} \cup C_{Collective}) \neq \emptyset$	Subkind, Role, Phase

O **padrão 33** contempla um DS composto de **Non-Sortals**. Nessa caso como os subtipos podem ser Rígidos, Semi-Rígidos e Anti-Rígidos e ainda podem ser tanto **Sortal** como **Non-Sortal,** a única exceção a ser realizada nesse caso é em relação a os **Substance Sortals** que não podem ser derivados.

O **padrão 34** tem a ocorrência de um Sortal Rígido no DS. Como o subtipo de um **Sortal** só pode ser outro **Sortal**, restam somente os **Sortals** não provedores de identidade como opção para o padrão (i.e., **Subkind**, **Role** e **Phase**).

4.8 Padrões de Tipos Derivados por Especialização em OntoUML

4.7.1 Especializações Inválidas

Dois casos genéricos de especializações inválidas são considerados preliminarmente aos casos mais específicos. O primeiro tipo de especialização inválida já citado em outros casos de derivação é a impossibilidade de ter um **Non-Sortal** como subtipo de um **Sortal**.

O segundo caso contempla a restrição do caso em que se tem um **Sortal** como tipo derivado por especialização de um **Non-Sortal.** Isso ocorre porque **Non-Sortals** são abstratos, e ainda, agrupam propriedades de **Sortals** distintos. Sendo assim, argumenta-se que os **Sortals** que são subtipos de um **Non-Sortal**, não devem ser determinados por uma derivação por especialização. Pelo contrário, **Non-Sortals** devem surgir como uma abstração de **Sortals**. Logo, tipos derivados por especialização satisfazer um dos dois casos. No primeiro tem-se

Non-Sortals como tipos derivados somente de **Non-Sortals** e no segundo **Sortals** como tipos derivados somente de **Sortals**.

4.7.2 Valores de Rigidez em Padrões de Tipos Derivados por Especialização

Tendo B como um tipo de objeto derivado por especialização de A:

Caso 1: Se A for Rígido ou Semi-Rígido então B pode ser Rígido ou Antí-Rígido. O valor quanto a rigidez é determinada de acordo com o conjunto de atributos em que a regra de especialização é aplicada. Se os atributos desse conjunto forem imutáveis então B é Rígido. Caso contrário deve existir um mundo m tal que uma instância x de B assuma valores para esses atributos de forma que x não satisfaça mais a condição de especialização e então B será Não-Rígido. Por exemplo, *Homem* é uma especialização Rígida de um tipo Rígido *Pessoa* porque o atributo gênero pode ser considerado imutável. Ou seja, um *Homem* x será *Homem* em qualquer mundo possível. Já um outro atributo como *idade* é mutável e faz com que as especializações de *Pessoa* aplicadas em relação a esse atributo sejam Anti-Rígidas. Por exemplo, um *Adolescente* é uma especialização Anti-Rígida de *Pessoa* que determina a condição "*Pessoa* entre 12 e 18 anos" uma vez que um *Adolescente* era uma criança e possivelmente se tornará um adulto.

Caso 2: Se A for Anti-rígido então o mesmo serve para B. Isso porque um tipo Anti-Rígido só pode ter como subtipo outro tipo Anti-Rígido.

4.7.3 Geração dos Padrões de Tipos Derivados por Especialização com base nas Combinações Válidas de Estereótipos em OntoUML

A **Tabela 4.11** apresenta os padrões para o caso 1:

Tabela 4.11 - Padrões de Tipos Derivados por Especialização de OntoUML para o Caso 1

Padrão	Estereótipo do Tipo A	Estereótipo do Tipo B
35	Category	Category, Role Mixin
36	Mixin	Category, Role Mixin
37	Kind	Subkind, Phase
38	Subkind	Subkind, Phase

Para o **padrão 35** e para o **padrão 36** pode-se chegar aos resultados eliminado os estereótipos inválidos para o tipo derivado. Inicialmente os **Substance Sortals** não podem ser derivados. **Roles**, **Phases** e **Subkinds** são **Sortals** que não podem ser derivados por especialização de **Non-Sortals**. Logo sobram **Category** e **Role Mixin** como opções válidas.

O **padrão 37** tem um **Kind** como tipo base. Nesse caso se o seu atributo for imutável seu subtipo será Rígido e então terá que ser um **Subkind**. Se for mutável então será um **Phase** (uma vez que o atributo do tipo A é um aspecto intrínseco, o que elimina a chance de ser um Role). O mesmo raciocínio serve para o **padrão 38** em que se tem um **Subkind** como tipo base.

A **Tabela 4.11** apresenta os padrões para o caso 2:

Tabela 4.12 - Padrões de Tipos Derivados por Especialização de OntoUML para o Caso 2

Padrão	Estereótipo do Tipo A	Estereótipo do Tipo B
39	RoleMixin	RoleMixin
40	Role	Phase
41	Phase	Phase

O **padrão 39** cobre o caso em que temos um **Role Mixin** como tipo base de uma derivação por especialização. Uma vez que seu subtipo deve ser **Non-Sortal** Antí-Rígido o único valor possível é um **Role Mixin**.

Para os **padrões 40 e 41**, o tipo derivado será um **Phase** porque condição de especialização caracteriza um aspecto intrínseco.

4.8 Padrões de Tipos Derivados por Participação em OntoUML

Tipos derivados por participação em OntoUML obedecem duas possíveis estruturas apresentadas na **Figura 4.9**. A primeira delas é chamada de bidirecional e ocorre quando se tem dois tipos bases que ao se relacionar geram dois tipos derivados por participação Anti-Rígidos. A segunda forma é quando se tem um tipo derivado por participação unidirecional. Ou seja, somente um dos tipos bases gera um tipo derivado ao se relacionar com o outro tipo base.

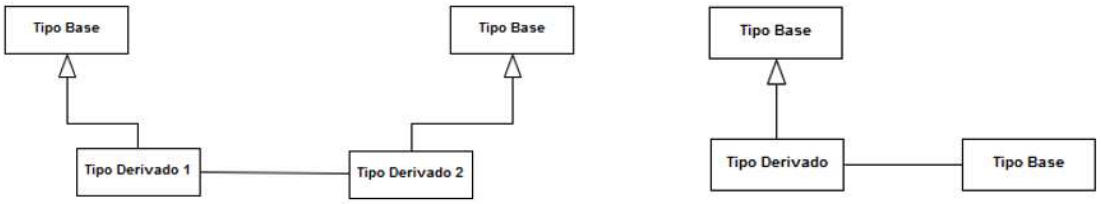

Figura 4.9 – Estruturas de Tipos Derivados por Participação esquerda (a) Bidirecional - direita (b) Unidirecional

Em uma grande parte dos casos de derivação por participação bidirecional em OntoUML sua estrutura final é equivalente ao padrão Relator apresentado em (Ruy et al. 2015). Nesse padrão os **Sortals** são os tipos bases enquanto os **Roles** são os tipos derivados.

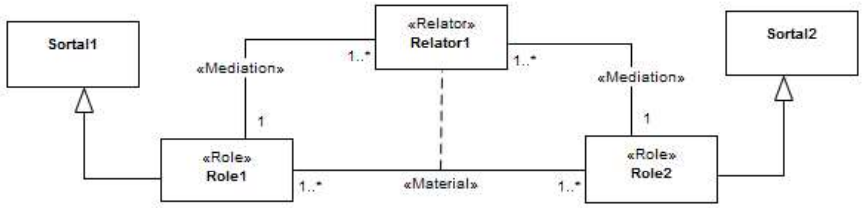

Figura 4.10 – Padrão Relator

Padrão 42: Tipos Derivados por Participação Bidirecional (Padrão Relator)

Uma outra forma de ocorrência de tipos derivados por participação em OntoUML se dá em relações todo parte tanto do todo para parte como da parte para o todo. Um *Barco Portador de Motor* é um *Barco* que é composto de *Motor*. Nem todo *Motor* é um *Motor de Barco*, logo *Motor de Barco* é também um **Role** de *Motor*. Isso evita o problema da cardinalidade opcional (i.e. cardinalidade mínima igual a 0), apresentado em (Bodart et al. 2001), do *Motor* em relação ao *Barco Portador de Motor*. Nesse exemplo ocorre tipos derivados por participação todo parte bidirecional.

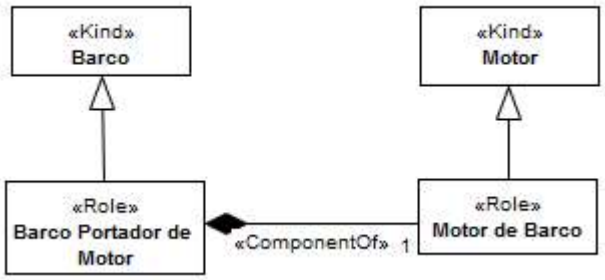

Figura 4.11 – Tipos Derivados por Participação Todo Parte Bidirecional

Um outro exemplo possível tem-se *Membro de Equipe* como um papel de uma *Pessoa* que compõe uma *Equipe*. Nesse caso temos um tipo derivado por participação todo parte Unidirecional.

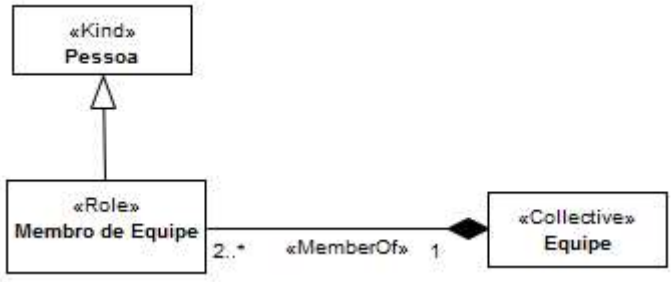

Figura 4.12 - Tipo Derivado por Participação Todo Parte Unidirecional

A aplicação de um padrão de tipo derivado por participação seja unidirecional ou bidirecional sempre terá um **Role** (quando unidirecional) ou dois (quando bidirecional) **Roles** como tipos derivados. Isso porque um tipo derivado por participação é definido em função de uma dependência externa (D+) o que é suficiente para categoriza-lo como **Role**.

Além disso o *Derivation Set* desse caso especial de derivação serão sempre formados por dois tipos bases que devem ser **Sortals**. Relações todo parte podem ser: (i)*memberof* (se a parte for um complexo funcional, **Quantity** ou **Collective** e o todo for um **Collective** como na Figura 4.12); (ii)componentOf (se for entre dois Complexos Funcionais como na Figura 4.11); (iii)subQuantityOf(se for entre dois Quantities); (iv)subcollectionOf (entre dois Collectives).

Padrão 43: Tipos Derivados por Participação Todo Parte Bidirecional

Padrão 44: Tipo Derivado por Participação Todo Parte Unidirecional

4.9 Padrões de Tipos Derivados por Especialização Passada em OntoUML

A especialização passada de um tipo C compreende todas as instâncias que já foram de um tipo base B em um mundo passado. Logo ambos, o tipo derivado e o tipo base, devem ser Não-Rígidos e subtipos de um mesmo supertipo A. Mais especificamente, o tipo derivado deverá ser obrigatoriamente Anti-Rígido uma vez que obrigatoriamente em um mundo passado todas as intâncias d o tipo derivado não eram suas instâncias.

Apresenta-se a seguir três formas possíveis de representação para um tipo de objeto derivado por especialização passada.

No primeiro caso, utiliza-se uma regra temporal porque ela é a única forma de se saber as instâncias de C. Por exemplo, se a única forma de se saber que um indivíduo x é um *Atleta* fosse sabendo seu nível de *atividade mensal*, então só conseguiremos determinar a população de *Ex-Atleta* através de um regra temporal aplicada em todos mundos passados. Ou seja, se em algum dos mundos passados, x atingiu um nível de *atividade mensal* de *Atleta* e depois foi desclassificado como tal, esse indíviduo então seria considerado um *Ex-Atleta*.

No segundo caso, pode-se ter um tipo derivado por especialização passada surgindo como supertipo de tipos já definidos no modelo. Nesse caso podemos representar a derivação sem a necessidade da regra temporal. Por exemplo, um *Ex-Casado* é alguém que é ou *Divorciado* ou *Viúvo*. Nesse caso, se quisermos ter um único tipo que representa o tipo derivado por especialização passada, esse tipo seria supertipo desses outros tipos do modelo. Ou seja, no exemplo dado, *Ex-Casado* é supertipo de *Divorciado* e *Viúvo*.

A última possibilidade é quando se tem uma especialização passada de um *Role*. Nesse caso pode-se optar por representar o tipo derivado por especialização passada com base em uma dependência externa passada. Nesse caso, não será preciso a regra temporal. Por exemplo, as instâncias do tipo Aluno tem uma *Matrícula* (**Relator**) ativa. Pode-se então particionar o **Relator** *Matrícula* entre *Matrícula Ativa* e *Matrícula Inativa*. Dessa forma os *Ex-Alunos* possuirão pelo menos uma *Matrícula Inativa* e nenhuma *Matrícula Ativa* como pode-se ver na Figura 4.13.

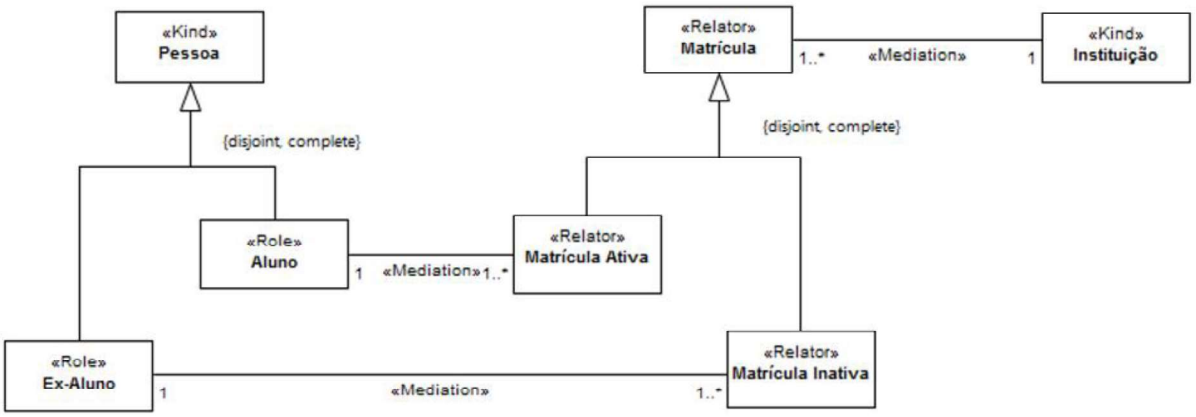

Figura 4.13 - Exemplo de Especialização Passada

Considerando os estereótipos OntoUML, existem então quatro variações para o tipo base possíveis que são os únicos quatro tipos de **Substantials** Não-Rígidos de OntoUML:

Padrão 45: TipoBase(Role) → TipoDerivado(Role)

Quando tratamos de um tipo que é a especialização passada de um **Role** esse é definido em função de uma dependência externa aplicada em um mundo passado. Por exemplo, *Ex-Aluno* é um **Role** definido em função de uma matrícula já vencida. Sendo assim, uma vez que esse tipo é definido em função de uma dependência externa (passada) ele deve ser um **Role**.

Padrão 46: TipoBase(Phase) → TipoDerivado(Phase)

Phase Partitions são determinados por partes complementares de um deteminado tipo. Isso quer dizer que os *generalization sets* que tem como subtipos **Phases** devem ser complete (i.e. todas as instâncias do supertipo devem ser instância de um dos **Phases** do Phase partition). Além disso, **Phase Partitions** são compostos somente por **Phases**. Logo a especialização passada de um **Phase** deve compor um Phase Partition juntamente com o tipo base.

Padrão 47: TipoBase(RoleMixin) → TipoDerivado(RoleMixin)

Um **Role Mixin** agrupa tipos Anti-Rígidos que herdam de *Substance Sortals distintos*. Logo, um tipo derivado por especialização passada de um tipo base **Role Mixin**, agrupa os tipos que

são a especialização passada dos subtipos agrupados pelo tipo base. Sendo assim, o tipo derivado só pode ser outro **Role Mixin**.

Padrão 48: TipoBase(Mixin) → TipoDerivado(RoleMixin)

Um Mixin agrupa tipos Anti-Rígidos que herdam de *Substance Sortals distintos*. Logo, um tipo derivado por especialização passada de um tipo base Mixin, agrupa os tipos que são a especialização passada dos subtipos anti-rígidos agrupados pelo tipo base. Sendo assim, o tipo derivado só pode ser outro RoleMixin.

5 SUPORTE FERRAMENTAL PARA UTILIZAÇÃO DE PADRÕES DE TIPOS DE OBJETOS DERIVADOS

5.1 Introdução

Este capítulo apresenta o suporte ferramental para padrões de tipos de objetos derivados em OntoUML. Além disso, o capítulo apresenta o mapeamento dos padrões de OntoUML para OWL-DL. Com o suporte ferramental apresentado neste capítulo, o modelador pode utilizar os padrões de tipos de objetos derivados em OntoUML, propostos no capítulo 4, de forma automática, garantindo, também, uma maior qualidade nos modelos gerados.

Como visto até aqui, nesta dissertação, o uso de objetos derivados é complexo, quando comparado a estruturas básicas de modelagem conceitual. Assim, o principal objetivo da implementação dos padrões é fornecer suporte ao modelador na construção de estruturas consistentes de padrões de tipos de objetos derivados. Ao permitir o uso dos padrões de forma automática, visa-se diminuir ainda mais o grau de inconsistências dos modelos em OntoUML. Isso ocorre, em especial, porque a implementação foi feita reutilizando uma infraestrutura existente, i.e. o editor OLED, que tem nele embutidos ferramentas que permitem a verificação sintática e semântica dos modelos, bem como sua validação por meio de simulação (geração de instâncias). Além disso, uma vez construído o modelo OntoUML, a automação realizada nesta dissertação permite que o modelador realize transformações automáticas de modelos OntoUML para OWL-DL, preservando a semântica dos padrões.

5.2 O Editor OLED

O OntoUML Lightweight Editor (OLED) disponibiliza um conjunto de recursos que, juntos, podem ser utilizados para desenvolver, avaliar e implementar ontologias em OntoUML. O objetivo da ferramenta é dar suporte ao processo de engenharia de ontologia promovendo a produção de modelos ontológicos consistentes. A ***Figura 5.1*** resume os recursos da ferramenta.

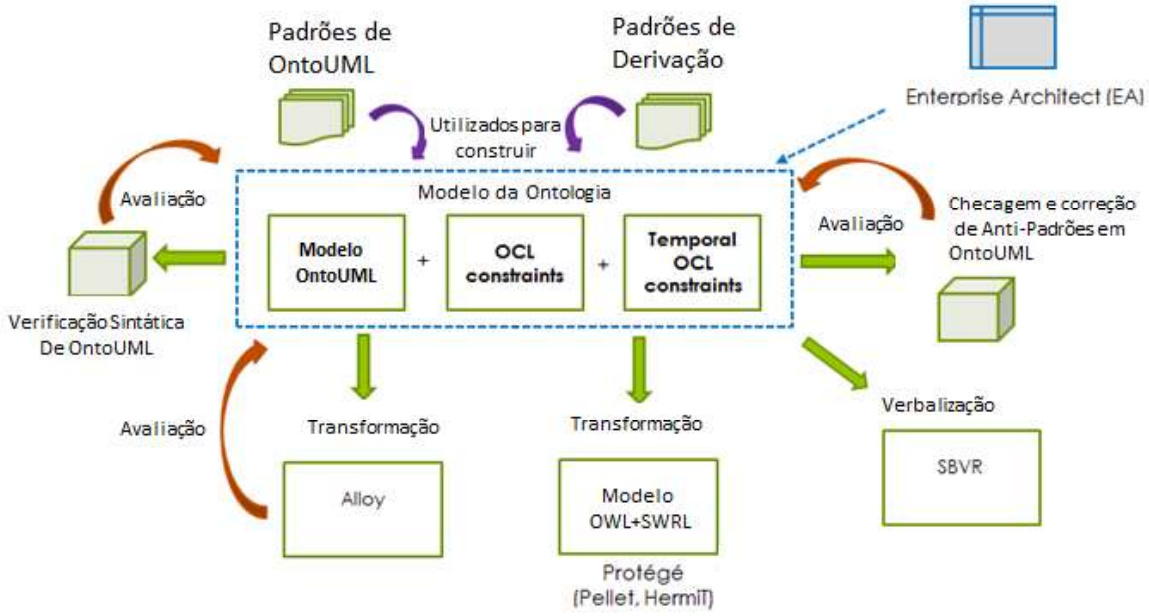

Figura 5.1- Recursos da Ferramenta OLED

Como mostra a figura 5.1, usando o OLED, é possível criar um modelo de Ontologia, composto por um Modelo OntoUML acrescido de axiomas descritos na linguagem OCL que podem ou não considerar aspectos temporais. Para auxiliar a construção do modelo OntoUML, o modelador pode aplicar padrões OntoUML assim como os padrões de derivação apresentados neste trabalho.

Três recursos podem ser aplicados para avaliação e correção de possíveis inconsistências do modelo das seguintes maneiras: (1) fazendo a checagem da ocorrência de anti-padrões da linguagem em um determinado modelo e aplicando correções automáticas, quando necessário; (2) transformando o modelo para a linguagem *Alloy* e simulando mundos possíveis e, a partir disso, fornecendo pistas ao modelador para deixar o modelo mais consistente, a partir do exame de instâncias indesejáveis; e (3) aplicando uma verificação sintática do modelo em relação ao metamodelo de OntoUML.

Por fim, a ferramenta dispõe de recursos de importação e exportação. Modelos OntoUML desenvolvidos na ferramenta *Enterprise Architect* podem ser importados na ferramenta sem qualquer perda de semântica. Além disso, depois de desenvolvido e avaliado, o modelo OntoUML pode ser transformado em uma ontologia operacional OWL-SWRL (Barcelos et al. 2013) para, por exemplo, ser utilizado como apoio a sistemas computacionais baseados em

ontologia, ou verbalizado no padrão SBVR (Guerson et al. 2015), para facilitar a comunicação com os interessados.

5.2.1 Aplicação de Padrões de Derivação no Editor OLED

Existem duas maneiras de se aplicar um padrão de tipos de objetos derivados durante a construção de um modelo. A primeira delas é construir toda a estrutura do padrão, ou seja, definir todos os tipos que constituem o *derivation set* (DS) e o tipo derivado de uma só vez. Essa estratégia, chamada neste trabalho de Derivação com a Estrutura Completa, é o foco da seção 5.2.1. A segunda forma é selecionar um DS com elementos que já fazem parte do modelo. A seção 5.2.2 trata dessa segunda estratégia, aqui chamada de Derivação em Tipos Previamente Definidos.

Na Derivação com a Estrutura Completa, o DS sempre será consistente (i.e., a derivação será válida), porque o padrão irá restringir somente os casos contemplados por padrões consistentes. Já no caso de Derivação com Tipos Previamente Definidos, será necessário validar se o modelador selecionou um DS válido para o tipo de derivação desejado. A seguir apresentam-se as formas de se aplicar os padrões na ferramenta.

5.2.2 Aplicando Padrões de Derivação com a Estrutura Completa

Para que o modelador possa criar a estrutura completa de um determinado padrão, disponibiliza-se a paleta de padrões de derivação como demonstrado na Figura 5.2.

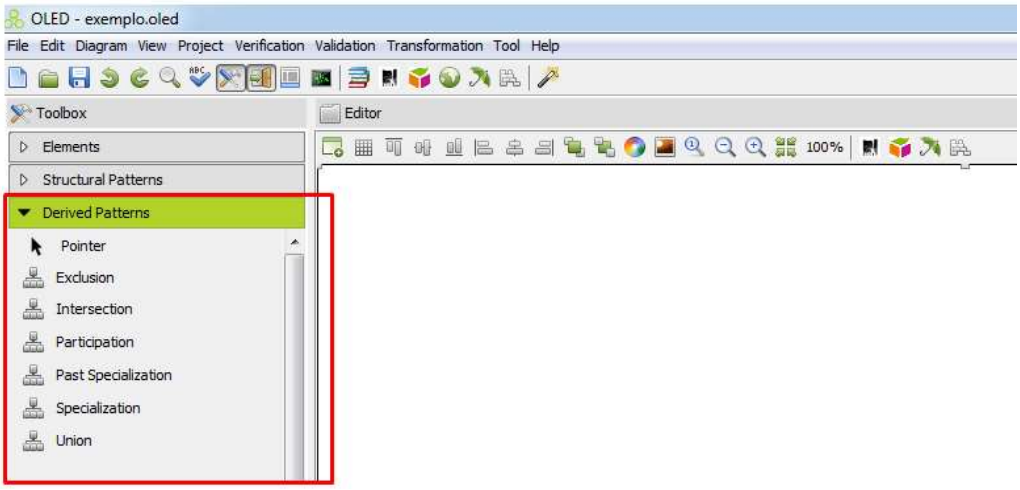

Figura 5.2-Paleta de Padrões de Derivação na Ferramenta OLED

Como demonstração, utiliza-se a aplicação de um padrão de derivação por exclusão (padrão **Exclusion**, **Figura 5.3**) em que *Desempregado* é definido como a exclusão de *Empregado*. Na primeira janela da **Figura 5.3** , pode-se perceber que uma vez que o tipo *Empregado* é definido como um **Role**, o seu supertipo pode ser qualquer outro estereótipo. Na segunda janela, percebe-se que uma vez definido *Pessoa* como um **Kind**, o estereótipo do tipo derivado é automaticamente inferido como **Role**[2]. Como anteriormente citado, pode-se perceber que não é possível criar uma estrutura inválida porque as seleções de estereótipos OntoUML são guiadas de forma a restringir os casos inválidos.

[2] Todos os padrões mencionados neste capítulo são definidos e explicados no capítulo 4.

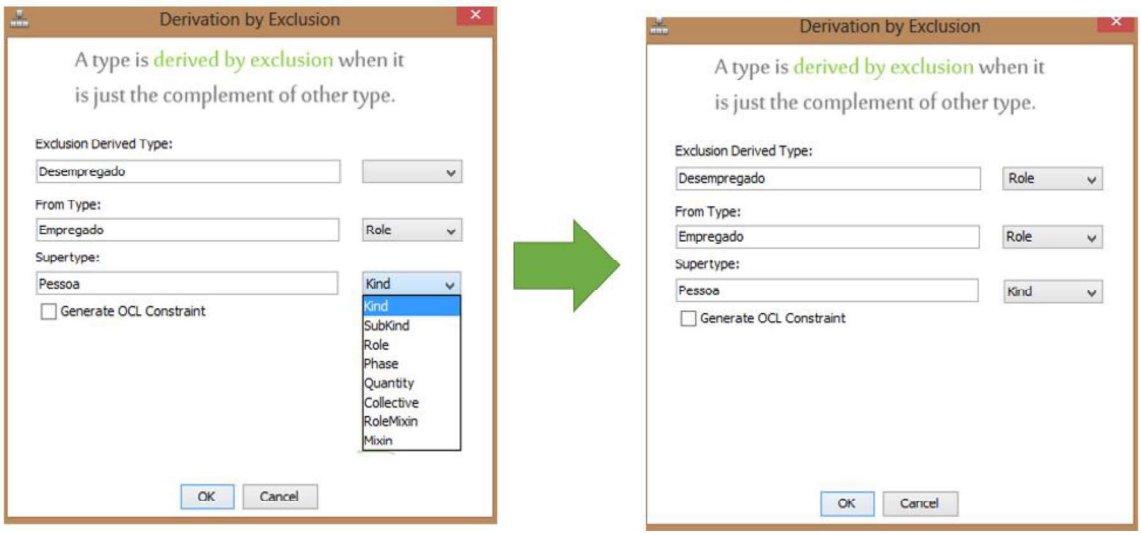

Figura 5.3 -Exemplo de Aplicação de Padrão de Derivação na Ferramenta OLED

5.2.3 Aplicando Padrões de Derivação em Tipos Previamente Definidos

Para aplicar um padrão de derivação em tipos já definidos, o modelador deve selecionar os elementos que fazem parte do DS e escolher o tipo de derivação. Dessa forma, o modelador pode vir a selecionar um DS inconsistente. Por essa razão, deve-se verificar a consistência da seleção realizada pelo modelador antes de aplicar o padrão. Além disso, quanto mais completo estiver o modelo, maior será a precisão da aplicação de cada padrão. Por modelo completo entende-se um modelo que satisfaz todas as regras estabelecidas pela linguagem OntoUML. Por exemplo, se for aplicada uma derivação por união em um DS composto de dois **Roles**, o tipo derivado por união poderia ser tanto outro **Role** como um **Role Mixin**. Se o modelo parcial estiver completo, ou seja, nesse caso se os supertipos dos **Roles** e provedores de princípio de identidade (i.e., ***Substance Sortals***) já estiverem definidos, será possível determinar se o tipo derivado é um **Role** (caso sejam subtipo do mesmo Substance Sortal) ou um **Role Mixin** (caso sejam subtipos de **Substance Sortals** distintos).

Por fim, na **Figura 5.3** pode-se ver o resultado da aplicação do padrão na ferramenta OLED. O tipo derivado por união, nesse caso, será automaticamente identificado como um **Role Mixin,** porque os tipos que compõem o DS são subtipos de ***Substance Sortals*** distintos., exemplifica em três passos, a aplicação do padrão na ferramenta OLED. Nesse exemplo,

supõe-se a necessidade de criar um tipo Cliente, que seria derivado pela união dos tipos *Cliente Pessoa Física* e *Cliente Pessoa Jurídica,* categorizados *como* **Role***:*

Passo I: Para aplicar o padrão na ferramenta OLED, o modelador deve selecionar os tipos do DS que, neste exemplo, são *Cliente Pessoa Física* e *Cliente Pessoa Jurídica* e clicar no botão direito do mouse. Então será exibida a paleta demonstrada da **Figura 5.3 – passo 2**;

Passo II: No passo 2, o modelador deve clicar em *Derive By* e depois selecionar o tipo de derivação que deseja aplicar que nesse caso será União (*Union*);

Passo III: No passo 3, será exibida uma janela para que o modelador dê um nome para o tipo derivado.

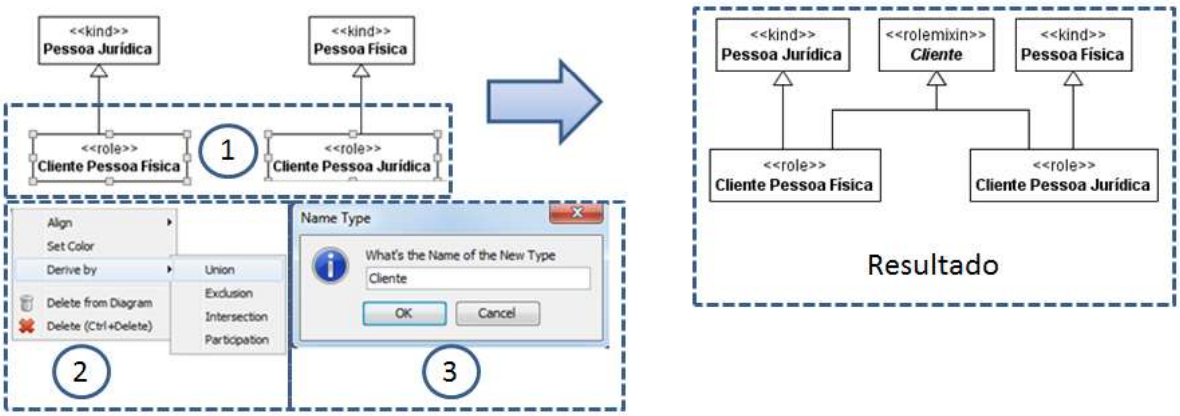

Figura 5.3 -Exemplo de Padrão de Derivação por União

Por fim, na **Figura 5.3** pode-se ver o resultado da aplicação do padrão na ferramenta OLED. O tipo derivado por união, nesse caso, será automaticamente identificado como um **Role Mixin,** porque os tipos que compõem o DS são subtipos de ***Substance Sortals*** distintos.

5.3 DETECTANDO PADRÕES DE DERIVAÇÃO NO EDITOR OLED

Na seção anterior, exemplificou-se o uso dos padrões propostos, mostrando que o OLED determina automaticamente, para o modelador, o estereótipo em OntoUML para o tipo derivado. É importante compreender como a ferramenta procede para definir qual o tipo

correto. Caso um padrão derivado proposto seja detectado, um conjunto de processos é executado, realizando as verificações necessárias para a detecção do estereótipo do tipo derivado para que o modelo fique correto. Além disso, é preciso que esses processos identifiquem os casos inválidos de derivação, para informar ao modelador qual foi a regra de OntoUML violada. Assim, ao construir um modelo em OntoUML, o modelador terá apoio automático para evitar erros quanto aos padrões de derivação propostos. A seguir, as seções 5.3.1 a 5.3.7 explicam os processos de detecção dos padrões para cada tipo de derivação implementado na ferramenta.

5.3.1 Processo de Detecção de Padrões de Tipos Derivados por União

O processo de detecção dos padrões de derivação por união é apresentado na figura 5.5.Nesse processo, são realizadas verificações necessárias para identificar o padrão a ser aplicado. Essas verificações são aplicadas em relação às meta-propriedades de OntoUML.

O OLED segue o processo definido pela **Figura 5.4**, realizando cada verificação até que se possa sugerir um estereótipo de OntoUML a ser utilizado para o tipo derivado. Por exemplo, na primeira verificação do processo (Qual o valor de Rigidez dos tipos do DS?), suponha o caso de que todos os tipos sejam rígidos (o fluxo segue para a esquerda). Em primeiro lugar, checa-se (Existe algum Substance Sortal ou Non-Sortal no DS?). Em caso negativo, verifica-se se (Os Membros do DS são subtipos do mesmo Substance Sortal?). Em caso afirmativo, o estereótipo sugerido é *Subkind*.

Como já discutido na 5.2.3, não seria possível realizar todas as verificações porque, em muitos casos, tem-se fragmentos de modelos incompletos quanto às regras de OntoUML. Por exemplo, um tipo categorizado como **Role** deve, obrigatoriamente, ser um subtipo de um ***Substance Sortal*** (**Kind, Collective e Quantity**). Porém, por questão de modularidade, por exemplo, o modelador pode achar útil criar um modelo incompleto que não satisfaça essa regra. Nesse caso, realiza-se apenas as verificações possíveis e, por consequência, sugere-se ao modelador todos os estereótipos possíveis, deixando a cargo dele decidir qual desses estereótipos deve ser escolhido.

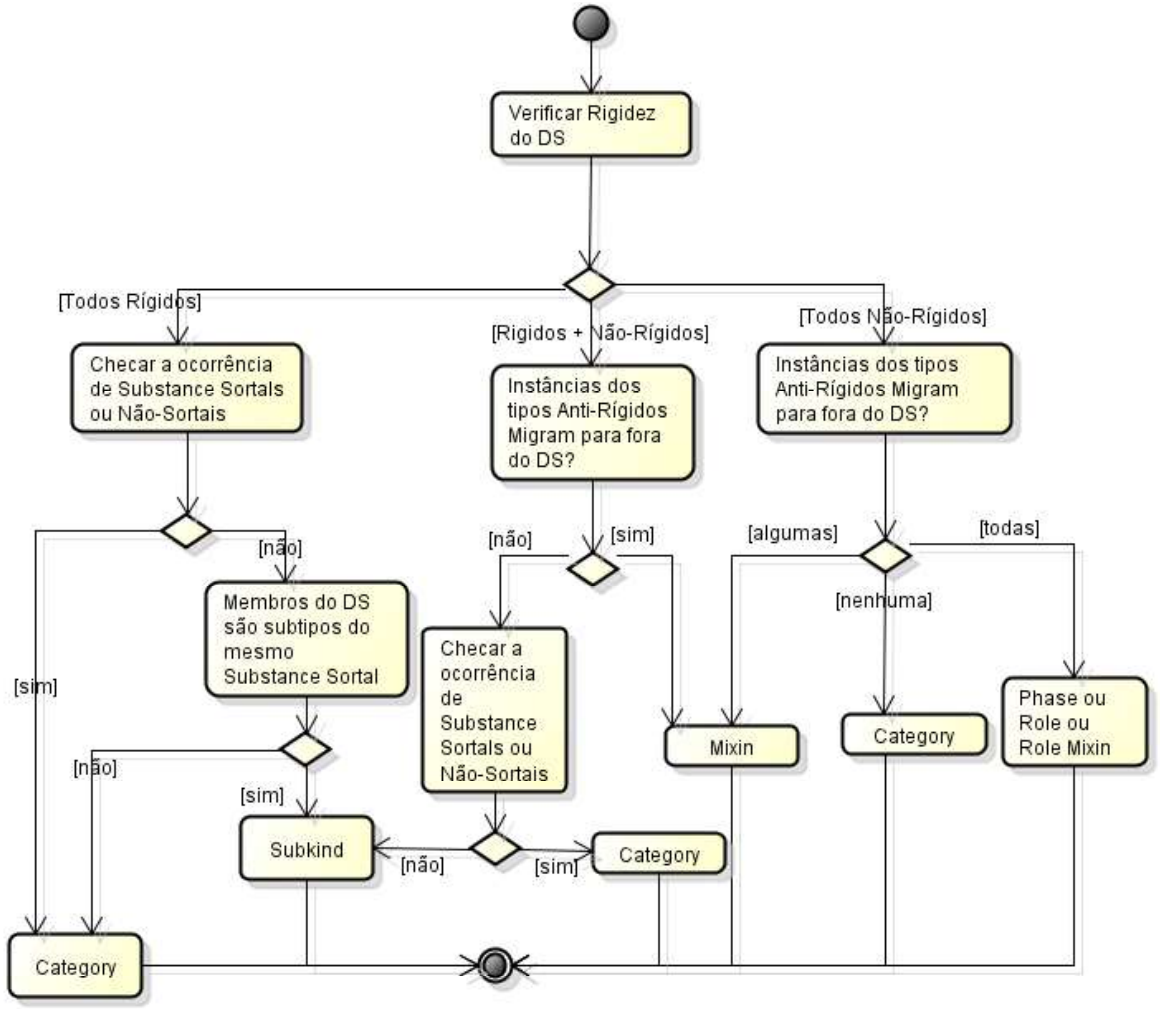

Figura 5.4- Processo de Detecção dos Padrões de União

No processo da **Figura 5.4**, somente duas das verificações são possíveis em todos os casos. A primeira delas verifica o valor de Rigidez dos tipos do DS (Qual o valor de Rigidez dos tipos do DS?). A segunda é a checagem quanto a existência de um Substance Sortal ou um Non-Sortal (Existe algum Substance Sortal ou Non-Sortal no DS?*)*. Para exemplificar o que ocorre no caso de um fragmento incompleto, considere o exemplo anterior, em que verificou-se que todos os tipos são Rígidos e que nenhum dos membros do DS são ***Substance Sortal***, O próximo passo é verificar se os (Membros do DS são subtipos do mesmo ***Substance Sortal?)***. Suponha, entretanto, que o fragmento do modelo é incompleto e, por essa razão, tal verificação não é possível. Nesse caso, há duas possibilidades de estereótipo para o tipo derivado: **Subkind** e **Category**. Assim, caberá ao modelador definir qual categorização deve ser selecionada.

5.3.2 Processo de Detecção de Padrões de Tipos Derivados por Exclusão

Para a detecção do padrão de derivação por exclusão, primeiramente verifica-se se o tipo base é um Sortal (O tipo base é um Sortal?). Caso seja, verifica-se a validade do padrão a ser aplicado verificando se esse tipo é um Substance Sortal (O tipo Base é um Substance Sortal?). Por exemplo, ao tentar aplicar uma derivação por exclusão em um **Kind**, a exclusão será invalidada e uma mensagem de erro será exibida para o modelador. Caso o padrão aplicado seja válido, então se aplica o mapeamento dos estereótipos do DS com os padrões de exclusão apresentados no Capítulo 4 de forma a se capturar o padrão a ser aplicado. A **Figura 5.5** apresenta o processo para tipos derivados por exclusão.

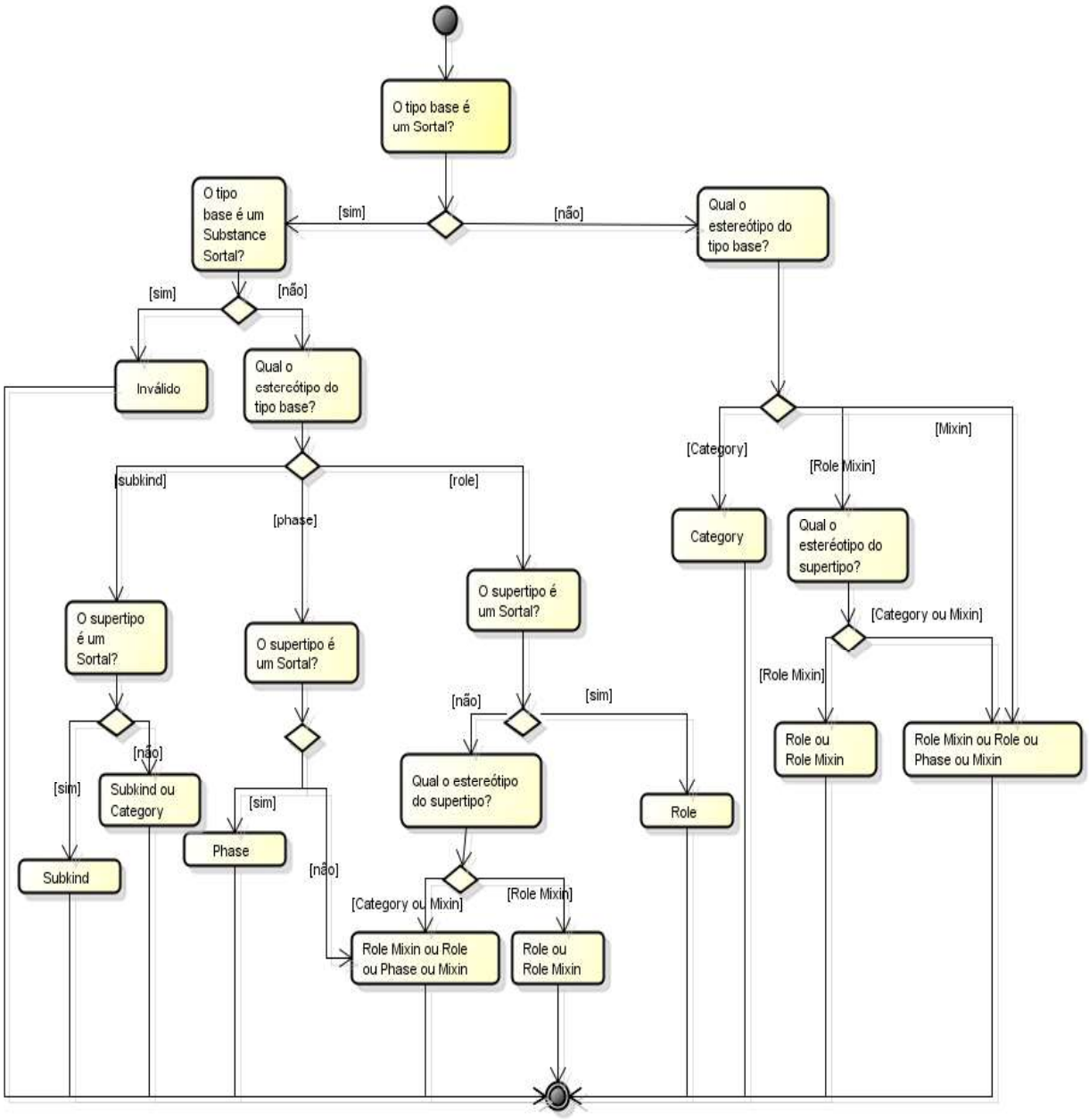

Figura 5.5 - Processo de Detecção dos Padrões de Exclusão

5.3.3 Processo de Detecção de Padrões de Tipos Derivados por Interseção

A Figura 5.6 apresenta o processo de detecção dos padrões de tipos derivados por interseção.

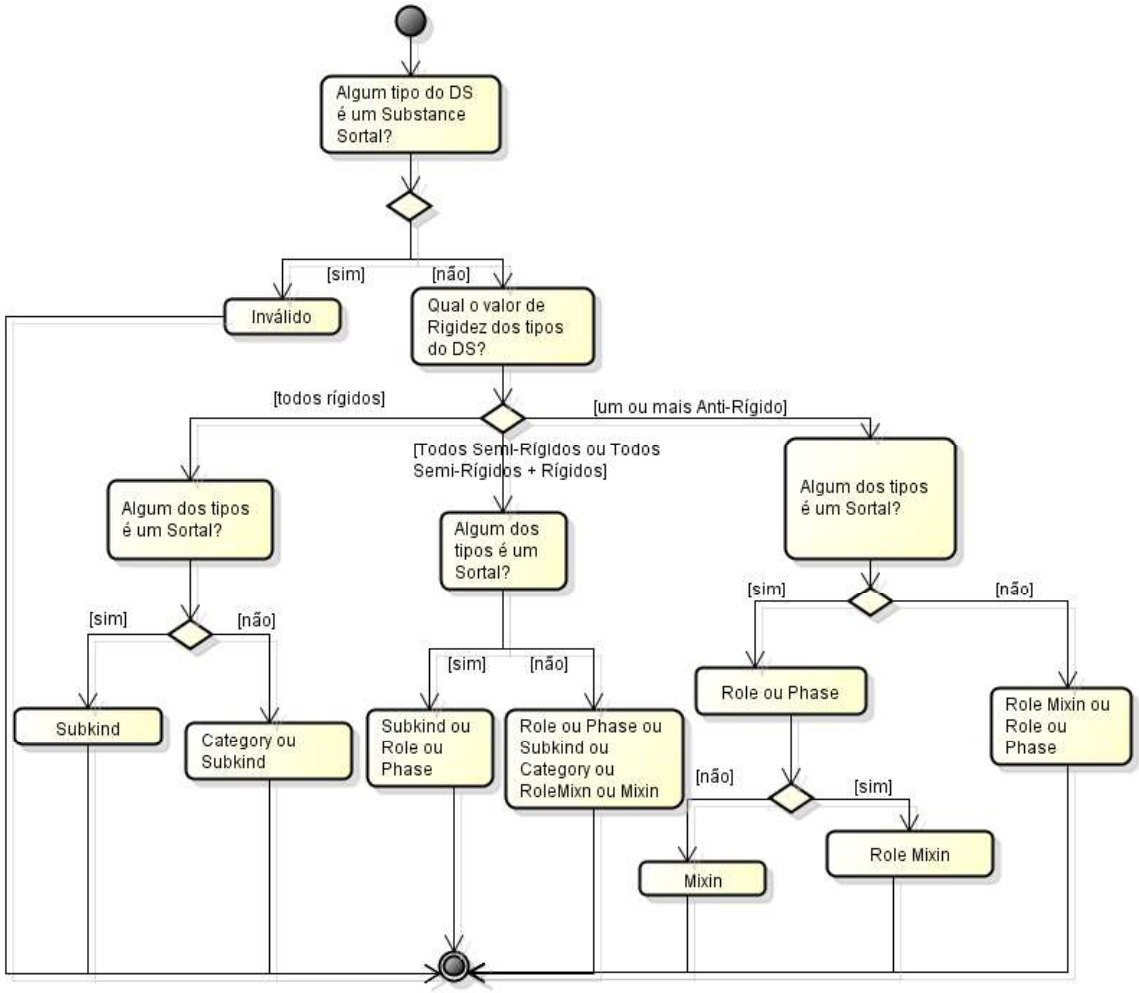

Figura 5.6 - Processo de Detecção de Padrão de Tipo Derivado por Interseção

Como mencionado na seção 4.6, a ocorrência de um ***Substance Sortal*** no DS torna uma interseção inválida (atividade "Inválida" no diagrama da **Figura 5.6**). A próxima verificação diz respeito a saber a rigidez dos tipos do DS (Qual o valor de Rigidez dos tipos do DS?). Em casos em que todos sejam **Rígidos,** então o tipo derivado será ou um **SubKind** ou um **Category**. Se existir algum membro do DS Anti-Rígido então o tipo derivado será Anti-Rígido também. Se houver pelo um **Sortal** no DS então o tipo derivado por interseção será ou um **Role** ou um **Phase**. Caso não hajam Sortals no DS o tipo derivado poderá ser um **Role Mixin**, um **Role** ou um **Phase**. Por fim, caso o DS tenha uma parte rígida e outra semi-rígida então

caso não hajam **Sortals** no DS então qualquer estereótipo será válido com excessão dos **Substance Sortals**. Caso contrário o tipo derivado terá que ser um **Sortal (Subkind, Role ou Phase)**.

5.3.4 Processo de Detecção de Padrões de Tipos Derivados por Especialização

A primeira verificação a ser realizada no processo de detecção de padrões de tipos derivados (**Figura 5.7**) por especialização é quanto à meta-propriedade relativa ao princípio de identidade (O tipo base é um Sortal?). Além dessa, a única verificação que refina a detecção dos padrões por especialização é a mutabilidade dos valores dos atributos em que a condição de especialização é aplicada. A verificação sobre a mutabilidade dos atributos não é possível de ser realizada automaticamente. Sendo assim, o processo limita-se a restringir especializações e generalizações inválidas. Cabe, então, ao modelador aplicar as regras de mutabilade contempladas no processo.

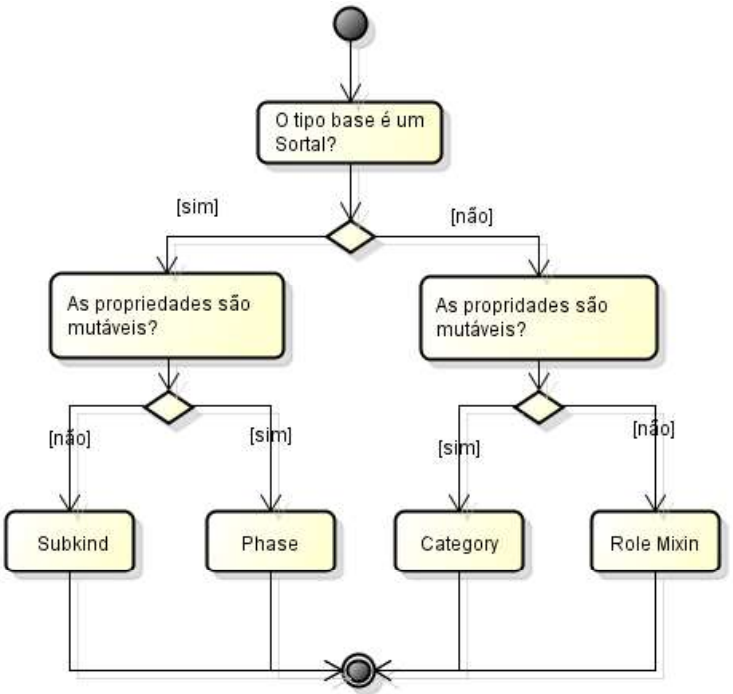

Figura 5.7 - Processo de Detecção dos Padrões de Tipos Derivados por Especialização

5.2.2. Processo de Detecção de Padrões de Derivação por Participação

Para tipos de objetos derivados por participação, a grande maioria das verificações a serem realizadas são resolvidas por decisões do modelador. A **Figura 5.8** apresenta o processo.

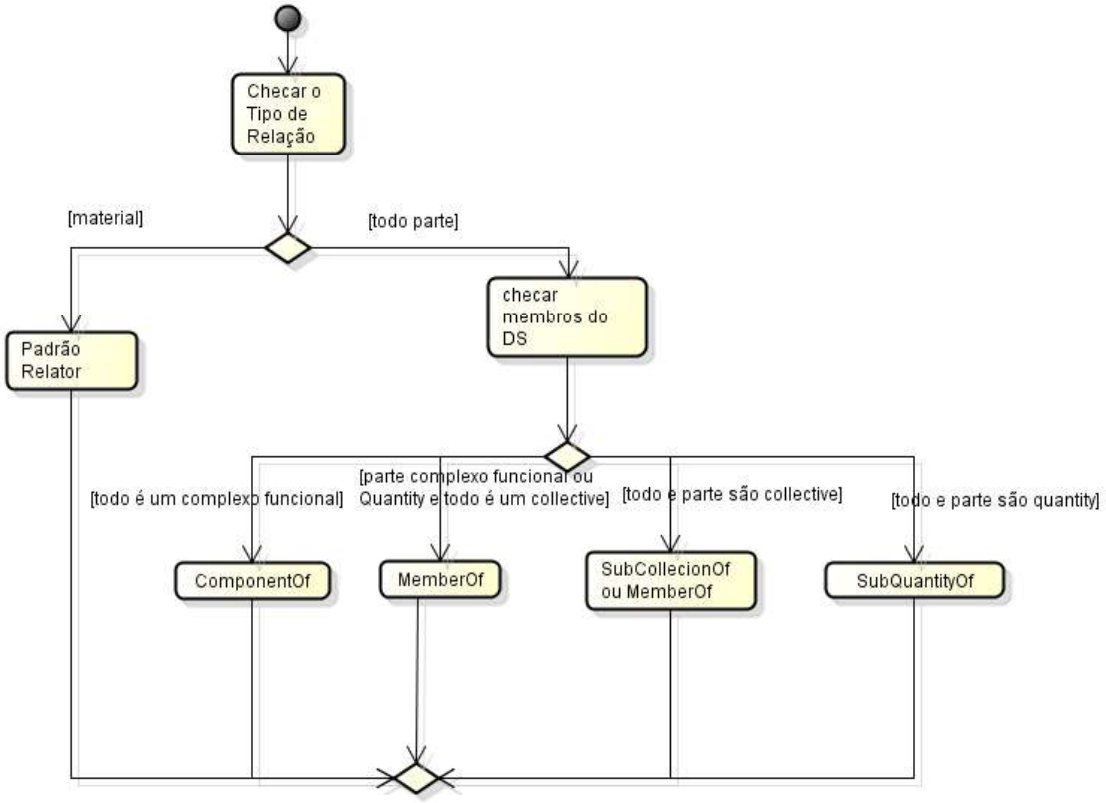

Figura 5.8 - Processo de Detecção dos Padrões de Tipos Derivados por Participação

Segundo o processo, o modelador deve selecionar os membros do DS e informar se a derivação a ser aplicada será unidirecional ou bidirecional. Além disso, o modelador precisa informar se a relação entre os tipos do DS será todo parte ou material (pergunta "Qual o tipo da relação entre os tipos base?" na **Figura 5.8**).

Se a relação for todo parte, a única verificação automática a ser realizada é a inferência sobre o tipo de relação de acordo com as categorizações dos tipos envolvidos: i) se o DS for formado

por dois **Collective** como parte e como todo, tem-se a relação ou ***memberOf*** ou ***subcollectionOf***; ii) se o todo for um complexo funcional, então a relação será de ***componentOf***.; iii) se o DS for formado por um complexo funcional ou Quantity como parte e por um **Collective** como todo, tem-se a relação ***memberOf***; iv) por fim, tendo-se um **Quantity** como parte e um **Quantity** como todo, tem-se uma relação de ***subQuantityOf***. Já se a relação entre os tipos do DS for material (i.e., mediada por um relator) então o padrão relator é aplicado.

5.3.5 Processo de Detecção de Padrões de Tipos Derivados por Especialização Passada

Por fim, a derivação por especialização passada refletirá sempre o estereótipo do tipo em que a derivação foi aplicada, conforme os padrões apresentados no Capítulo 4. A **Figura 5.9** apresenta o processo de detecção.

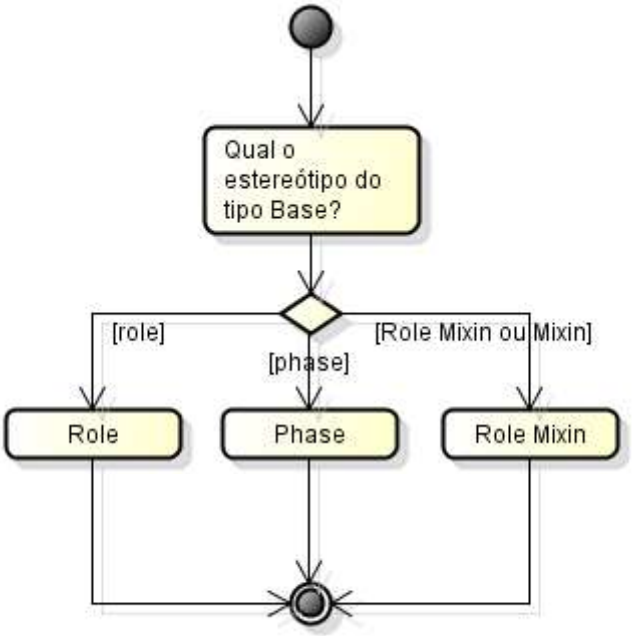

Figura 5.9 - Processo de Detecção dos Padrões de Tipos Derivados por Especialização Passada

Como mostra a figura 5.10, nesse caso, a única verificação realizada é (Qual o estereótipo do tipo base?). Dada a resposta, o tipo derivado segue o estereótipo do tipo base.

5.4 Representação das Regras de Derivação

A ferramenta OLED provê um editor para a criação de regras OCL do modelo. Quatro dos tipos de derivação envolvem a geração de regras para expressar a semântica do padrão. Neste trabalho, foi implementada a geração das regras automaticamente para tipos de objetos derivados por Exclusão, Interseção e Especialização Passada. As regras de tipos derivados por especialização não são geradas automaticamente porque elas não seguem uma estrutura comum.

5.4.1 Geração Automática de Regra OCL de Tipos de Objetos Derivados por Exclusão

Ao aplicar um padrão de tipos de objetos derivados por exclusão na ferramenta, poderá ser acrescentada de forma automática a regra de exclusão OCL junto ao modelo. A regra de exclusão sempre respeitará a estrutura do exemplo da **Figura 5.10.**

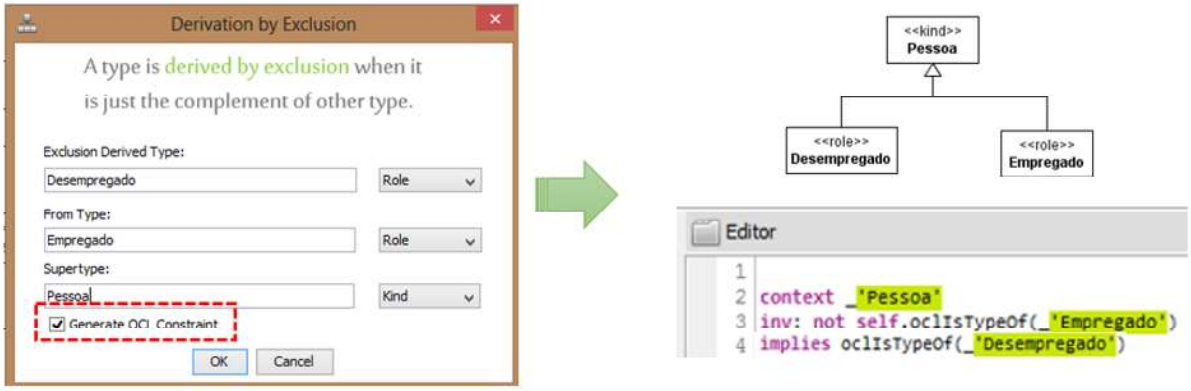

Figura 5.10 - Geração da Regra OCL de Exclusão

Como mostra a **Figura 5.10**, para tipos de objetos derivados por exclusão criados com o modelador pode optar por gerar ou não a regra de exclusão. A regra gerada no exemplo expressa que qualquer indivíduo classificado como *Pessoa* que não for *Empregado* será automaticamente classificado como *Desempregado*.

5.4.2 *Geração Automática de Regra OCL de Tipos de Objetos Derivados por Interseção*

A aplicação de um padrão de tipos de objetos derivados por interseção na ferramenta produzirá automaticamente a regra OCL que faz com que o tipo derivado tenha suas instâncias como a interseção total dos tipos do DS. A **Figura 5.11** apresenta um exemplo em que o DS é formado por um **Subkind (***Órgão Municipal***)** e um **Category** (*Entidade Fiscalizadora*). A aplicação de uma interseção entre esses tipos para gerar um **Subkind (***Órgão Municipal Fiscalizador***)** gera a regra que faz com que todo individuo que seja instância tanto de *Órgão Municipal* quanto de *Entidade Fiscalizadora* seja também uma instância do tipo derivado *Órgão Municipal Fiscalizador*.

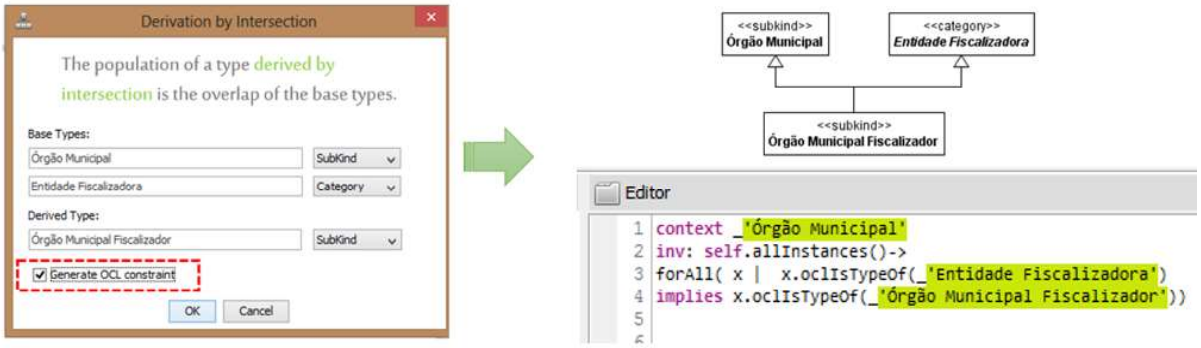

Figura 5.11 - Geração da Regra OCL de Interseção

5.4.3 *Geração Automática de Regra OCL de Tipos de Objetos Derivados por Especialização Passada*

Para padrões de tipos derivados por especialização passada, a regra OCL gerada considera aspectos temporais. No exemplo da **Figura 5.12**, um *Ex-Atleta* é um tipo derivado por especialização passada de *Atleta.* A regra OCL temporal expressa que para todos os mundos possíveis (World), um indivíduo de Ex-Atleta deve ter sido em um mundo passado (allPrevious) um *Atleta* e não deve ser um *Atleta* no mundo corrente.

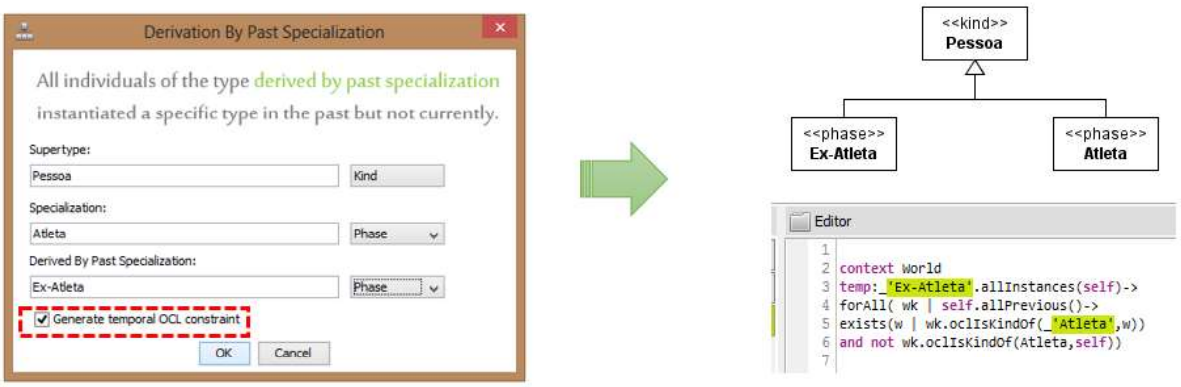

Figura 5.12 - Geração da Regra OCL Temporal de Especialização Passada

5.3. Classes derivadas em OWL

A linguagem OWL é amplamente utilizada para codificar ontologias operacionais, ou seja, aquelas que darão suporte a sistemas, por exemplo, na Web Semântica. Por essa razão, investiga-se, neste trabalho, a representação dos padrões de tipos de objetos derivados como classes derivadas em OWL. Propõe-se uma extensão da transformação automática de OntoUML para OWL implementada na ferramenta OLED, acrescentando o código das classes derivadas, para representar adequadamente os padrões propostos. Considera-se a especificação de OWL 2 (W3C 2012), utilizando-se um perfil de OWL baseado em *Description Logic* denominado OWL-DL.

5.4.4 Classes Derivadas por União e Interseção em OWL-DL

OWL-DL provê construtos específicos para expressar Classes que são União e Classes que são Interseção de outras Classes. Para Classes que representam a União de outras Classes, OWL-DL tem as propriedades para Classes não disjuntas (***union of***) *e* ***disjoint union of*** para classes disjuntas, enquanto interseções são representadas com a propriedade ***intersection of***. Sendo assim, neste trabalho, implementou-se a transformação automática de cada ocorrência de padrões de derivação por união e interseção de OntoUML para o construto equivalente em OWL-DL. Por padrão, uma interseção em OWL representa uma interseção total. A criação de interseções parciais necessita de axiomas adicionais sobre a definição da Classe derivada.

A **figura 5.11** apresenta exemplos da transformação automática de OntoUML para OWL-DL sendo aplicada em duas estruturas que compreendem um padrão de tipo derivado por união e um padrão de tipo derivado por interseção. Para demonstrar o mapeamento, utiliza-se a sintaxe de RDF para o modelo OWL.

Figura 5.11 – Mapeamento de OntoUML para OWL-DL – União e Interseção

5.3.2. Classes Derivadas por Exclusão em OWL-DL

Para representar uma classe derivada por exclusão em OWL-DL, basta-se definir uma classe como equivalente a negação de outra classe. Porém, esse axioma não terá nenhum resultado na maioria dos raciocinadores automatizados, pois eles assumem a premissa de mundo aberto. Porém, Grimm & Motik (2005) criaram um raciocinador, que considera aspectos de mundo fechado para OWL. Em função disso, neste trabalho, decidiu-se por manter o axioma de exclusão em padrões de tipos derivados por exclusão. Mais especificamente, em Description Logic a exclusão de um tipo A com base em um tipo B é representada como $A \equiv \neg B$. A **Figura 5.13** demonstra a criação de uma classe B como exclusão de uma classe A na ferramenta Protegé e o código em OWL-DL na sintaxe de RDF.


```
<owl:Class rdf:about="#A">
    <owl:equivalentClass>
        <owl:Class>
            <owl:complementOf rdf:resource="#B"/>
        </owl:Class>
    </owl:equivalentClass>
</owl:Class>
```

Figura 5.13 -Classe Derivada por Exclusão em OWL-DL

5.3.3. Classes Derivadas por Especialização em OWL-DL

Um tipo derivado por especialização é transformado para OWL com a mesma estrutura de Classe e Subclasse. Além disso, quando a condição de especialização está descrita em OCL, é transformada para OWL-DL.

Em OCL, é possível descrever a condição de especialização, tanto em termos de uma regra de integridade como em termos de uma regra de derivação. Por exemplo, para expressar a condição de especialização que *Pessoas* do gênero masculino são *Homens*, pode-se definir uma regra de integridade "*Homens* devem ser pessoa do gênero masculino". Essa mesma regra pode ser descrita como uma regra de derivação da seguinte forma: "Se o gênero da *Pessoa* é masculino, essa *Pessoa* é um *Homem*". Condições de especialização em OCL são descritas como invariantes. A seguir apresentam-se duas formas de se escrever essa condição de especialização:

I) context Homem

inv: self.genero = ‘masculino’

II) context Pessoa

inv: self.genero = ‘masculino’ implies self.oclIsTypeOf(Homem)

Embora as duas regras sejam descritas como invariantes, pode-se perceber que o segundo caso trata-se de uma regra de derivação, uma vez que ela estabelece as condições necessárias para que alguém seja classificado como Homem. Em ambos os casos, o mapeamento para OWL-DL é realizado em termos de uma restrição aplicada à classe OWL que representa a classe derivada por especialização, como exemplificado a seguir:

```
<owl:Class rdf: #Tipo_Derivado_por_especialização">
      <owl:Restriction>
        <owl:onProperty rdf:resource="#dataproperty" />
        <owl:allValuesFrom>
        <rdfs: Datatype>
        Especificar Valores de derivação
        </rdfs: Datatype>
        </owl: tipo de restrição>
      </owl:Restriction>
</owl:Class>
```

O fragmento de código OWL acima representa uma restrição sobre a classe derivada. A restrição será aplicada em função de um *dataproperty*, que equivale a um atributo em OWL-DL. De acordo com a condição de especialização, determinam-se os valores que tornam o indivíduo instância da classe. Por exemplo, se um *Adolescente* for definido como alguém que tenha entre 11 e 17 anos, é possível definir a especialização da seguinte forma:

```
<owl:Class rdf: #Adolescente">
      <owl:Restriction>
        <owl:onProperty rdf:resource="#idade " />
        <owl:allValuesFrom>
        <rdfs: Datatype>
                <owl: onDataRange rdf:resource=”&xsd;nonNegativeInteger” />
                <xsd:minInclusive rdf:datatype=”&xsd;int”> 11</xsd:minInclusive>
                <xsd:minInclusive rdf:datatype=”&xsd;int”> 17</xsd:minInclusive>
        </rdfs: Datatype>
        </owl: tipo de restrição>
      </owl:Restriction>
</owl:Class>
```

Embora a regra seja descrita em termos de uma restrição, na grande maioria dos raciocinadores, ela é interpretada também como regra de derivação. Ou seja, se tivermos um indivíduo instanciado como Pessoa com idade entre 11 e 17 anos, então esse indivíduo será classificado como adolescente.

5.3.4. Classes Derivadas por Participação em OWL-DL

Uma classe derivada por participação em *Description Logic* pode ser expressa através de uma equivalência: ClasseDerivada ≡ =nR.ClasseImagem. A fórmula expressa que os indivíduos da classe derivada serão aqueles que tiverem alguma relação R com a classe que é a imagem da relação. O n que precede a Relação é um número que expressa a cardinalidade da relação do lado da Imagem.

Para exemplificar a transformação de um padrão de tipo derivado por participação, a **Figura 5.12** apresenta o exemplo de casamento, em que tanto o marido como a esposa podem ser considerados tipos derivados por participação, por participarem de um casamento. Para o Marido, a fórmula em *Description Logic* expressa anteriormente ficaria da seguinte forma: Marido ≡ =1R.Casamento. Na Figura 5.utiliza-se o Protegé como editor OWL para facilitar a visualização.

Figura 5.2 - Exemplo de Transformação de Tipos Derivados por Participação

Como se pode perceber no diagrama da esquerda da **figura 5.12**, o Relator *Casamento* une *Marido* a *Esposa*. A relação material derivada do casamento *é casado com* e serve para ambos. Após a transformação, pode-se perceber que a descrição da Classe *Marido* é dada por exatamente uma (cardinalidade do lado da Imagem) relação inversa de *une Esposa a*

(**INV.une_Esposa_a**) porque as relações de mediação sempre partem do *Relator* (i.e. tem o **Relator** como Domínio). O mesmo acontece com a relação material *e_casado_com*. Percebe-se então que a semântica do tipo derivado por participação foi mantida em OWL-DL.

6. CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Este trabalho aborda a construção de padrões de criação de tipos de objetos derivados para OntoUML. A distinção de tipos de objeto como base ou derivado justifica-se somente sob uma perspectiva específica na qual se tem uma visão meramente lógica dos tipos de objeto de um modelo. Sob essa visão, são considerados derivados os tipos de objeto que, por meio de regras lógicas explicitas, estabelecem as condições suficientes para que um indivíduo seja considerado uma de suas instâncias.

Embora esses tipos sejam definidos com base em propriedades extensionais, os padrões aqui propostos não se restringem à criação de tipos definidos unicamente por propriedades extensionais. Isso porque é possível ter uma visão estritamente focada em aspectos extensionais de tipos de objetos que são essencialmente intencionais. Ou seja, um mesmo tipo de objeto pode ser visto sob diferentes perspectivas. Em uma dessas perspectivas, pode-se considerar somente as propriedades extensionais desse tipo de objeto e, nesse caso, ele será entendido como um tipo de objeto derivado.

A aplicação dos padrões de tipos de objetos derivados em OntoUML facilita a criação de fragmentos de modelo consistentes. Pode-se então, utilizar a teoria aqui fundamentada, para guiar o processo de criação de tipos, com base em outros tipos já definidos no modelo. Para isso, foi utilizada uma análise que considera as propagações das meta-propriedades de OntoUML para os padrões de criação de novos tipos com base em operações de conjuntos. Ao entender os padrões de tipos derivados de OntoUML dessa forma, é possível utilizá-los como um método automático de construção do modelo. Assim, basta que o modelador defina os tipos base do modelo e aplique os padrões para gerar os demais tipos. Além disso, os padrões de tipos derivados de OntoUML tornam mais claros alguns aspectos complexos do processo de modelagem em OntoUML, no que diz respeito as decisões de qual estereótipo aplicar a um certo conceito.

Como resultado principal deste trabalho, apresenta-se todas as possíveis formas de se criar um tipo de objeto derivado, considerando as meta-propriedades ontológicas de OntoUML. De

forma geral, os padrões apresentados são facilmente justificáveis com base nas regras definidas por OntoUML. Em alguns casos específicos, não foram encontrados argumentos suficientes para determinar, de forma precisa, a categorização do tipo de objeto derivado. Esses casos podem ser fruto de duas possíveis causas. A primeira delas é que existe a possibilidade de que argumentos, baseados em outras propriedades não consideradas neste trabalho, possam restringir um determinado padrão ou até torna-lo inválido. A segunda é que tais padrões, por desconsiderarem as propriedades intencionais dos tipos, podem vir a gerar um conceito possível do ponto de vista lógico, porém inconsistente ou inútil do ponto de vista ontológico.

Do ponto de vista da Engenharia de Ontologias, os padrões aqui apresentados se encaixam na fase conceitual, em decisões de *design* e na fase de implementação. Neste trabalho, dá-se seguimento a uma abordagem que busca aperfeiçoar a qualidade dos modelos conceituais, visando produzir especificações mais bem elaboradas como forma de se chegar a modelos de implementação mais ricos em semântica. Por isso, discutem-se meios de se transformar modelos ontológicos desenvolvidos em OntoUML para linguagens de ontologia operacionais. Nesse contexto, OWL-DL foi a linguagem de ontologia operacional escolhida por ser amplamente utilizada e por ser uma linguagem fortemente projetada para a definição de tipos com base em propriedades extensionais.

Este trabalho é parte de uma série de iniciativas que buscam automatizar o processo de modelagem em OntoUML. Dentre elas, destaca-se o trabalho de (SALES 2014), que elabora métodos de checagem de consistência de modelos com base em anti-padrões ontológicos. Destaca-se, ainda, o trabalho de (Santos 2015) e de (Guizzardi et al. 2011), que objetivam a criação de padrões OntoUML para facilitar a construção de ontologias nessa linguagem.

No que se refere a trabalhos futuros, duas perspectivas de trabalhos futuros são exploradas, de forma a dar continuidade ao trabalho aqui apresentado:

A primeira delas trata de expandir os padrões de tipos de objetos derivados para a teoria de Multilevel Theory (MLT) (Carvalho & Almeida 2015), que considera tipos de objeto em diferentes níveis de instanciação. Neste trabalho, trata-se somente de tipos de objetos derivados que ocorrem em um nível de instanciação específico, sem considerar os construtos específicos de MLT. Uma vez que os padrões de tipos de objetos derivados podem ser aplicados em uma

modelagem multinível, percebe-se a necessidade de uma análise aprofundada da forma apropriada de aplicar os padrões nesse contexto.

A segunda linha é a criação de métodos de propagação de padrões de tipos derivados em cascata. Para que se possa entender a proposta, apresenta-se um trecho de modelo em que tem-se duas formas de compra. Um cliente pode realizar compras em uma *Loja Virtual* ou em uma *Loja Física.* Ao aplicar um padrão de derivação por união nos tipos *Loja Física* e *Loja Virtual*, cria-se o tipo genérico *Loja*. Pode-se então, propagar essa operação para os outros elementos da estrutura, com o intuito de ter uma estrutura genérica de *Compra* como mostra a **Figura 6.1**. Ao desenvolver os métodos de propagação dessas operações, pode-se criar camadas de abstração baseadas em operações pré-definidas, otimizando-se ainda mais o processo de construção de modelos OntoUML.

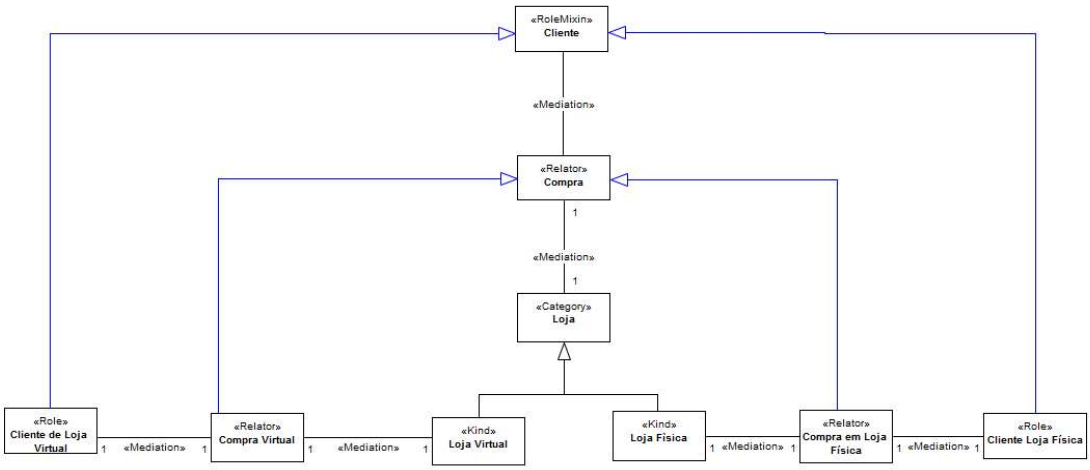

Figura 6.1 - Proposta de Propagação dos Padrões de Tipos Derivados em OntoUML

REFERÊNCIAS

Abiteboul, S. & Hull, R., 1987. IFO: A Formal Semantic Database Model. *ACM Trans. Database Syst.*, 12(4), pp.525–565. Available at: http://doi.acm.org/10.1145/32204.32205.

Anon, 2014. OMG Unified Modeling LanguageTM (OMG UML), Infrastructure.

Barcelos, P.P.F. et al., 2013. An Automated Transformation from OntoUML to OWL and SWRL. *Ontobras*, pp.130–141. Available at: http://ceur-ws.org/Vol-1041/ontobras-2013_paper44.pdf.

Barcelos, P.P.F. et al., 2013. An Automated Transformation from OntoUML to OWL and SWRL. In *ONTOBRAS*. Citeseer, pp. 130–141.

Berners-Lee, T., Hendler, J. & Lassila, O., 2001. The Semantic Web. *Scientific American*, 284, pp.34–43. Available at: http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21.

Bodart, F. et al., 2001. Should Optional Properties Be Used in Conceptual Modelling? A Theory and Three Empirical Tests. *Information Systems Research*, 12(4), pp.384–405.

Carvalho, V.A. De & Almeida, J.P.A., 2015. Towards a Well-Founded Theory for Multi-Level Conceptual Modelling. , pp.114–118. Available at: http://nemo.inf.ufes.br/mlt.

Evermann, J., 2002. Using UML for conceptual modeling.

Falbo, R. de A., 2011. SABiO: Systematic Approach for Building Ontologies. *Springer-Verlag Berlin Heidelberg*.

Grimm, S. & Motik, B., 2005. Closed World Reasoning in the Semantic Web through Epistemic Operators. In *OWLED*.

Guerson, J. et al., 2015. OntoUML Lightweight Editor: A Model-Based Environment to Build, Evaluate and Implement Reference Ontologies. In *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop*. IEEE, pp. 144–147.

Guizzardi, G., 2006a. On Ontology, ontologies, Conceptualizations, Modeling Languages, and (Meta)Models. *DB&IS, Frontiers in Artificial Intelligence and Applications*, 155, pp.18–39.

Guizzardi, G., 2005. *Ontological Foundations for Structural Conceptuals Models*. University of Twente.

Guizzardi, G., 2012. Ontological meta-properties of derived object types. *Advanced Information Systems Engineering*. Available at: http://link.springer.com/chapter/10.1007/978-3-642-31095-9_21 [Accessed August 25, 2013].

Guizzardi, G., 2006b. The role of foundational ontologies for conceptual modeling and domain ontology representation. In *Databases and Information Systems, 2006 7th International Baltic Conference on*. IEEE, pp. 17–25.

Guizzardi, G., Graças, A. & Guizzardi, R.S., 2011. Design Patterns and Inductive Modeling Rules to Support the Construction of Ontologically Well-Founded Conceptual Models in OntoUML. In C. Salinesi & O. Pastor, eds. *Advanced Information Systems Engineering Workshops SE - 44*. Springer Berlin Heidelberg, pp. 402–413.

Halpin, T., 2006. Object-role modeling (ORM/NIAM). In *Handbook on architectures of information systems*. Springer, pp. 81–103.

Halpin, T.A., 2007. Subtyping Revisited. In *Proceedings of the 12th Workshop on Exploring Modeling Methods for Systems Analysis and Design (EMMSAD 2007), held in conjunction with the 19th Conference on Advanced Information Systems (CAiSE 2007)*. pp. 128–138.

Halpin, T.A. & Bloesch, A.C., 1999. Data modeling in UML and ORM: a comparison. *J. Database Manag.*, 10(4), pp.4–13.

Hammer, M. & Mc Leod, D., 1981. Database description with SDM: a semantic database model. *ACM Transactions on Database Systems (TODS)*, 6(3), pp.351–386.

Lindland, O.I., Sindre, G. & Solvberg, A., 1994. Understanding quality in conceptual modeling. *Software, IEEE*, 11(2), pp.42–49.

Moody, D. & van Hillegersberg, J., 2009. Evaluating the visual syntax of UML: An analysis of the cognitive effectiveness of the UML family of diagrams. In *Software Language Engineering*. Springer, pp. 16–34.

Mylopoulos, J. & Schmidt, J., 2010. *Conceptual Modeling*, Morgan & Claypool.

Odell, J.J., 1998. *Advanced object-oriented analysis and design using UML*, Cambridge University Press.

Olivé, A., 2007. *Conceptual modeling of information systems*, Springer Science & Business Media.

Olivé, A., 1998. *Conceptual Modelling of Information Systems*, Available at: http://medcontent.metapress.com/index/A65RM03P4874243N.pdf [Accessed August 25, 2013].

OMG, 2010. OMG Unified Modeling Language TM (OMG UML), Infrastructure. , (May).

Ruy, F.B. et al., 2015. Ontology Engineering by Combining Ontology Patterns. In *The 34rd edition of the International Conference on Conceptual Modeling*.

SALES, T.P., 2014. *ONTOLOGY VALIDATION FOR MANAGERS*. UFES.

Santos, V.A. Dos, 2015. *Pattern-Based Approach For Building Ontological Conceptual Models*. Universidade Federal do Espírito Santo.

Simons, P.M., 1987. Parts: A study in ontology.

Vojislav, B. & Leon, J., 2000. Evaluating the quality of reference models. In *Conceptual Modeling—ER 2000*. Springer, pp. 484–498.

W3C, 2012. OWL Structural Specification. Available at: http://www.w3.org/TR/owl2-syntax/.

Zamborlini, V.C., 2011. Estudo de Alternativas de Mapeamento de Ontologias da Linguagem OntoUML Para OWL: Abordagens Para Representação de Informação Temporal. *Federal University of Espírito Santo. Available only in Portuguese*.