UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

DEPARTAMENTO DE INFORMÁTICA

MESTRADO EM INFORMÁTICA

ALEX PINHEIRO DAS GRAÇAS

SUPORTE AUTOMATIZADO PARA CONSTRUÇÃO DE MODELOS CONCEITUAIS BEM FUNDAMENTADOS

VITÓRIA, AGOSTO 2010

ALEX PINHEIRO DAS GRAÇAS

SUPORTE AUTOMATIZADO PARA CONSTRUÇÃO DE MODELOS CONCEITUAIS BEM FUNDAMENTADOS

Proposta de 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, AGOSTO 2010

DEDICATÓRIA

às tristezas, que me ensinaram a apreciar o riso.
às dificuldades que me ensinaram a perseverar.
às lutas, sem as quais não poderia existir a vitória.

aos que me amaram, estes são os que me tornaram algo.

aos que passaram pela minha vida e de alguma maneira me tornaram o que sou hoje.

à Deus, autor da vida, pois sem ele,
nada do que foi feito se fez.

AGRADECIMENTOS

Agradeço a minha **família**, pelo carinho, pelo amor, e pelos parâmetros que me ensinaram a andar. Neste momento que me preparo para começar uma, desejo que a minha seja tão bela quanto esta.
Ao meu **Irmão** pelas conversas, pela paciência e até aos raros desentendimentos.
À minha **Mãe** pelo carinho, conversas, atenção, amizade e amor.
Ao meu **Pai** pelo amor, paciência e esforço que auxiliaram a chegar onde estou hoje.
Aos meus **Amigos**, que me tiveram paciência comigo durante este período de mestrado.
Em especial ao **Paulo Sergio**, pelos auxílios e dicas valiosas.
Ao professor **Giancarlo** pela orientação.
À **FAPES/FUNCITEC** pelo apoio financeiro.
Aos inúmeros **Professores** de vida e instituições de ensino, que a vida colocou em meu caminho.

À minha N**oiva e futura Esposa Gabriela**, pela paciência neste período.

A **Deus**, pela força, sustento e graça.

RESUMO

A adoção de linguagens ontologicamente bem fundamentadas para construção de ontologias, apesar de vir ganhando mais adeptos, esbarra nas dificuldades da utilização desta linguagem por modeladores inexperientes. Este trabalho propõe um guia metodológico para apoiar as construções de ontologias utilizando a linguagem OntoUML Este guia auxilia o projetista no processo de modelagem através do estudo da estrutura da linguagem, suas restrições sintáticas e padrões de projetos. Além disso, as interações com o modelador na utilização deste guia são utilizadas para gerar um design rationale do processo de modelagem. Por final é apresentada uma implementação Web de uma ferramenta que demonstra as propostas apresentadas neste trabalho.

Palavras-chave: Ontologias, Design Rationale, Ontologias de Domínio, Reuso, Ontologia de Referência, UFO, Modelagem Conceitual.

ABSTRACT

The adoption of ontologically well founded languages for building ontologies has acquiring more adopters, however its widespread use stumble on using difficulties by inexperienced modelers. This dissertation proposes a methodological guide to support the building of domain ontologies using the ontologically well founded language OntoUML. This guide through the analysis of language structure, restrictions and design patterns helps the designer on the modeling process. Beyond, the building support interactions are used to generate a design rationale. At last, it is presented a Web application implementing the ideas proposed in this work.

Keywords: Ontologias, Design Rationale, Domain Ontologies, Reuse, Foundational Ontologies,UFO, Conceptual Model.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

CAPÍTULO 1. INTRODUÇÃO

Este capítulo apresenta as motivações que serviram de base para a elaboração desta dissertação e os objetivos que foram buscados. Também é abordada a maneira como a pesquisa foi conduzida e qual a estrutura que foi adotada para melhor compreensão deste trabalho.

1.1.MOTIVAÇÃO

A área de sistemas baseados em conhecimento tem galgado cada dia mais relevância. A cada dia aumenta a necessidade de que as máquinas possam não só armazenar, mas também compreender os dados. O processo de criar modelos que reflitam a realidade não é trivial, porém de extrema importância na atualidade, para que as máquinas possam atribuir significado à informação.

Na ciência da computação existe a divisão entre dados e informação(FRICKÉ, 2009). Pode-se definir dados como:

> *Dados são símbolos que representam propriedades de objetos, eventos e ambiente. São produtos de observação. Observar é medir. A tecnologia de instrumentação é altamente desenvolvida. (ACKOFF, 1989)*

Outro exemplo de captura de dados é quando necessitamos preencher um formulário. O próximo nível na hierarquia é Informação. A informação são dados relevantes. Quando perguntamos: onde alguém está?; Ou quando ele chegará?; obtemos uma informação. Pode-se definir informação como: dados que são processados para serem úteis; proveem respostas para as perguntas "quem", "o que", "onde" ou "quando". (ACKOFF, 1989)

Atualmente, através dos consórcios que definem os padrões da grande rede, a Web se desenvolve para um ambiente que contenha, do ponto de vista das máquinas não somente dados, mas que compreenda as informações e atribua significado aos dados para que as máquinas estejam aptas a processar o conteúdo presente na Web.

> *"Eu tenho um sonho para a Web, no qual computadores tornem-se capazes de analisar dados na Web – o conteúdo, links e transações entre computadores. Uma 'Web Semântica", a qual poderia fazer isto possível, ainda deve emergir, mas quando isto acontecer, os mecanismos do dia-a-dia de comércio,*

> *burocracia e nossas vidas diárias serão guiadas por máquinas falando com máquinas. Os agentes inteligentes que as pessoas têm promovido há épocas finalmente se materializarão."*
>
> *(BERNERS-LEE, 1999)*

A Web Semântica se tornou o próximo nível de desenvolvimento a ser atingido, e o termo "semântica" também se tornou expressão comum em revistas de informática. Grandes portais de buscas – como o Google, Bing, Yahoo – buscam melhorar seus resultados apresentando respostas com maior relevância para o usuário, tentando compreender o que o usuário pesquisa. Todo esse esforço passa por compreender como transformar dados em informação e como capturar e representar esta informação.

Usualmente a Inteligência Artificial tenta solucionar problemas definindo o conhecimento de forma estritamente funcional, tentando incorporar em uma base de conhecimento os passos que um *expert* tipicamente usaria. Contudo Clancey (1993) defende a *modeling view,* na qual a principal preocupação da Engenharia de conhecimento é modelar os sistemas no mundo e não replicar como as pessoas pensam. Deste modo, a atividade de modelagem do conhecimento deve ser feita de maneira tão independente quanto possível da tarefa de solução do problema, o objetivo deve ser modelar o domínio abordado. (GUARINO, 1995)

Gruber (1993) propõe que ontologias sejam utilizadas como forma de compartilhamento de conhecimento. Neste trabalho o autor define ontologia como a "uma especificação explícita de uma conceitualização compartilhada". Por esta razão torna-se claro que o estudo de ontologias pode beneficiar o processo de construção de conhecimento dos sistemas inteligentes. Esses argumentos demonstram a importância do desenvolvimento de modelos que representem fielmente o domínio proposto.

Desde a organização de comunidade de ontologias em meados da década de 1990, várias linguagens têm sido adotadas para a representação de ontologias. A partir do ano 2000, com a popularização do uso de ontologias como artefato de representação do conhecimento no contexto da Web Semântica, as linguagens de representação de ontologias propostas naquele contexto têm ganhado crescente notoriedade, com destaque para OWL (Ontology Web Language) (W3C, 2004) e RDF (Resource Description Framework)(W3C, 2004).

No entanto, como demonstrado em (GUIZZARDI, 2006), apesar de frequentemente possuírem propriedades computacionais de interesse para muitas aplicações (ex., decidibilidade, tratabilidade computacional), as chamadas ontologias leves (*lightweight ontologies*) construídas nestas linguagens não são modelos suficientemente ricos para dar suporte aos problemas de interoperabilidade esperados de ocorrer em um contexto como a Web.

Por exemplo, os relacionamentos expressos utilizando a linguagem OWL não capazes de dizer se a ligação entre os conceitos é um relacionamento é um relacionamento todo-parte; um tipo T em OWL não expressa se é possível um indivíduo instância deste tipo T deixar de ser instância. Deste modo, como advogado em (ibid.), o uso de linguagens como OWL e RDF deve ser complementado pelo uso de linguagens ontologicamente bem fundamentadas e que sejam ricas do ponto de vista de expressividade e comunicabilidade.

Para que as ontologias criadas atinjam seu objetivo de: apoiar a comunicação; facilitar o aprendizado de um domínio; e representem os domínios propostos com fidelidade, se fazem necessárias ontologias criadas com linguagens ontologicamente bem fundamentadas.

Uma linguagem construída especificamente para lidar com essa questão e adotada neste trabalho é a OntoUML(GUIZZARDI, 2005). Trata-se de uma versão estendida da UML[1] 2.0, cujo metamodelo incorpora primitivas de modelagem e uma rica axiomatização derivadas das teorias ontológicas que compõem a ontologia de fundamentação UFO(GUIZZARDI e WAGNER, 2005).

A linguagem OntoUML tem sido utilizada com sucesso na criação de ontologias em diversos domínios complexos, como eletrofisiologia do coração (GONÇALVES, ZAMBORLINI e GUIZZARDI, 2009) e Petróleo e Gás (GUIZZARDI, LOPES, *et al.*, 2010). Além disso, tanto a linguagem quanto seus conceitos subjacentes têm experimentado uma crescente adoção (HALPIN e MORGAN, 2008)(BAUMAN, 2009), muito embora tal linguagem apresente um alto grau de complexidade para modeladores iniciantes.

[1] No contexto deste trabalho quando referir-se a UML, trata-se do metamodelo do diagrama de Classes.

Logo, o que motiva a nossa pesquisa é que apesar de comprovada a importância do desenvolvimento de ontologias de domínio com linguagens ontologicamente bem fundamentadas, a utilização dessas linguagens é de difícil acesso para modeladores inexperientes. Portanto, o objetivo deste trabalho, como apresentado na próxima seção, é facilitar a adoção de linguagens ontológicamente bem fundamentadas na criação de ontologias de domínio.

1.2. OBJETIVOS E ESCOPO

O objetivo deste trabalho é facilitar o processo de criação de ontologias de domínio. Em um primeiro nível, o objetivo é tornar mais fácil a criação de ontologias utilizando uma linguagem ontologicamente bem fundamentada. A ontologia de fundamentação para a criação de modelos a qual será utilizado nesta dissertação é a UFO (*Unified Foundatinal Ontology*)(GUIZZARDI e WAGNER, 2004). Neste trabalho será utilizado um fragmento da OntoUML, baseado na UFO-A.

Para alcançar o objetivo deste trabalho propõe-se: (i) demonstrar que a OntoUML é uma linguagem de padrões e que, portanto, é possível prever certas estruturas sintáticas da linguagem; (ii) um guia metodológico para a criação de ontologias utilizando a UFO para apoiar o projetista na criação de modelos utilizando a linguagem OntoUML; (iii) inferir o estereótipo de um conceito representado em OntoUML de maneira indireta, de maneira que o modelador não tenha que conhecer os estereótipos da linguagem.

Além disso, as interações do usuário com o guia metodológico serão capturadas, para dessa maneira, explicar por que o modelo está projetado da maneira que está. Esta informação é conhecida como *Design Rationale*[2]. Esta informação é utilizada neste trabalho para criar sugestões para o modelador e apoiá-lo na tarefa de criar ontologias.

Com o intuito de demonstrar a usabilidade da metodologia, foi criada uma aplicação WEB que automatiza as propostas deste trabalho para auxiliar a criação de modelos conceituais por modeladores iniciantes.

---

[2] Segundo (LEE e LAI, 1996) *Design Rationale* é uma explicação de porque o artefato, ou parte do artefato, está projetado da maneira que está.

1.3. ABORDAGEM

Este trabalho foi realizado com uma abordagem interativa durante o processo de pesquisa e desenvolvimento. Primeiramente foi demonstrado que a OntoUML é uma linguagem de padrões de projetos, posteriormente foi demonstrado que estes padrões podem ser utilizados para criar regras de formação do modelo.

Para dialogar com o projetista e armazenar as interações com o modelador foi pesquisada a utilização do *Design Rationale*(MEDEIROS, 2006). O *Design Rationale,* foi utilizado para capturar as justificativas dos modeladores e melhorar a qualidade das sugestões apresentadas pela ferramenta.

Como passo seguinte foi construído um modelo para armazenar os padrões de projetos presentes em um modelo e o *Design Rationale* da sua construção. Este modelo é utilizado para armazenar dados históricos e apresentar sugestões durante a interação do usuário com o guia metodológico.

Como último passo de apoio ao modelador foram pesquisados padrões léxicos, que são padrões textuais utilizados na linguagem natural que torna possível capturar algumas metapropriedades dos conceitos. Estes padrões foram utilizados para apresentar frases ao modelador que possam refletir o domínio sendo modelado.

Por fim foi desenvolvida uma aplicação WEB para implementar as propostas deste trabalho e demonstrar a sua exequibilidade.

1.4. ORGANIZAÇÃO DO TRABALHO

Esta dissertação está organizada em seis capítulos. Além deste capítulo, que apresenta a Introdução, há mais cinco capítulos com o seguinte conteúdo:

- Pressupostos teóricos - apresenta um referencial teórico acerca dos assuntos utilizados no desenvolvimento desta dissertação. Abordam-se as definições, os tipos, as aplicações de ontologia, destacando a teoria da UFO, pois é a base para a proposta apresentada neste trabalho. Neste capítulo, também são discutidas as terminologias relevantes para a criação do sistema proposto EMOF(OMG, 2006) e OCL(OMG, 2006).
- Guia Metodológico para a Construção de Modelos Utilizando a OntoUML – Este capítulo apresenta um guia metodológico para a construção de ontologias

utilizando OntoUML e os padrões de projetos que emergem da UFO. Esses padrões são utilizados para criar diálogos que guiarão o modelador na tarefa de criação de uma ontologia. Também é apresentado o DPModel, um modelo para capturar informações no contexto de padrões de projetos presentes em um modelo. Estas informações são utilizadas para gerar sugestões para o usuário durante a interação com o guia metodológico.

- Suporte ao Modelador para Estereotipação de Conceitos em OntoUML – Este capítulo apresenta um método para aplicar os estereótipos da linguagem OntoUML nos tipos, sem a necessidade de conhecer as distinções ontológicas feitas por eles. Também é apresentada uma extensão do DPModel para armazenar o *Design Rationale* capturado com interação do usuário.
- Construção de uma ferramenta de apoio a criação de Ontologias de domínio – Este capítulo apresenta uma ferramenta para modelagem utilizando a OntoUML. Essa ferramenta utiliza os métodos de suporte proposto neste trabalho para facilitar o processo de criação de ontologias. Também se demonstra um caso de uso no qual são utilizadas as propostas deste trabalho para criar um modelo conceitual. Por fim é apresentada a arquitetura interna da ferramenta.
- Considerações Finais – Este capítulo apresenta as conclusões finais e trabalhos futuros.

CAPÍTULO 2. PRESSUPOSTOS TEÓRICOS

Neste capítulo, são apresentados os pressupostos teóricos, assuntos necessários para o desenvolvimento desta tese. É apresentado primeiramente uma discussão sobre ontologias e o processo de metamodelagem apresentando a OntoUML como uma linguagem Ontologicamente bem fundamentada. Também é abordado um Editor para a construção de modelos em OntoUML.

2.1. ONTOLOGIA

Pesquisas a respeito de Ontologia tem ganhado relevância em Ciências da Computação, um termo antigamente confinado a pesquisas filosóficas, ganhou áreas como Inteligência Artificial, Linguística Computacional, Teoria de Banco de Dados entre outras.

Ontologia é uma descrição, um modo de representar formalmente um domínio, a sua definição mais famosa na área de ciências da computação é uma especificação de uma conceitualização(GRUBER, 1993).

Áreas como Engenharia do Conhecimento, representação do conhecimento, modelagem de informação, integração de informação são exemplo de áreas que a sua importância já é reconhecida(GUARINO, 1998). Outro exemplo muito comum da utilização de ontologias é na criação de um vocabulário comum para comunicação(DEVEDZIC, 1999) ou para a gerência do conhecimento(GRUBER, 1993).

Um fator extenuante na gerência de conhecimento é que muitas vezes eles se encontram de maneira não estruturada. Usando ontologias, pode-se anotar informações semânticas em artefatos de informação não estruturados, visando assim a obtenção de resultados mais precisos em pesquisas de informação (Guimarães, 2002).

Apesar de tantas pesquisas o termo ontologia tem sido utilizado erroneamente para referenciar qualquer modelo conceitual construído para representar um domínio, por isso a importância de definir corretamente o que é uma ontologia.

2.1.1. DEFINIÇÕES

Etimologicamente a palavra vem do grego, onde "*ont"* é o presente particípio do verbo *einai* (ser), logo a palavra ontologia significa o estudo da existência. De acordo com (GUIZZARDI, 2005), Ontologia, com letra maiúscula, é uma ciência definida por ser um ramo da meta-física que estuda a natureza e as relações da existência e do ser. O seu estudo é iniciado por Aristóteles na Grécia antiga. Tem o intuito de ser uma ciência geral do *ser-qua-ser*. Porém este termo para designar este saber somente começou a ser utilizado no século XXVII pelos filósofos Rudolf Göckel e Jacob Lorhard.

O conceito de ontologia, com letra minúscula, é a especificação de conceitos de um determinado domínio, independente da linguagem que o domínio é representado. O estudo de ontologias atualmente passa por um momento ascendente graças a Web Semântica(GUIZZARDI, 2007).

Uma ontologia é considerada um conjunto de axiomas projetados para representar o significado de um vocabulário(GUARINO, 1998).

Uma ontologia é um termo usado para referenciar um entendimento compartilhado do mesmo domínio de interesse, que pode ser usado como um arcabouço para resolver problemas de comunicação, interoperabilidade, "reuso", especificação de sistemas. (USCHOLD e GRUNINGER, 1996)

2.1.2. NÍVEIS ONTOLÓGICOS

Em (GUARINO, 1998) é discutido sobre o nível de especificidade de um modelo ontológico, sendo proposta a seguinte classificação – ilustrada na Figura 1 – baseando-se no nível de especificidade da ontologia:

- Ontologias de fundamentação: descrevem conceitos gerais os quais são independentes de um domínio particular.
- Ontologias de Domínio ou de Tarefa: Descrevem respectivamente vocabulários relacionados a um domínio especifico (como medicina, ou direito) ou uma tarefa ou atividade genérica. Estas ontologias utilizam-se dos termos da ontologia de fundamentação.
- Ontologia de Aplicação descreve conceitos dependentes de um domínio particular e uma tarefa particular. Usualmente descreve os papéis

desempenhados por conceitos de uma ontologia de domínio em uma tarefa específica.

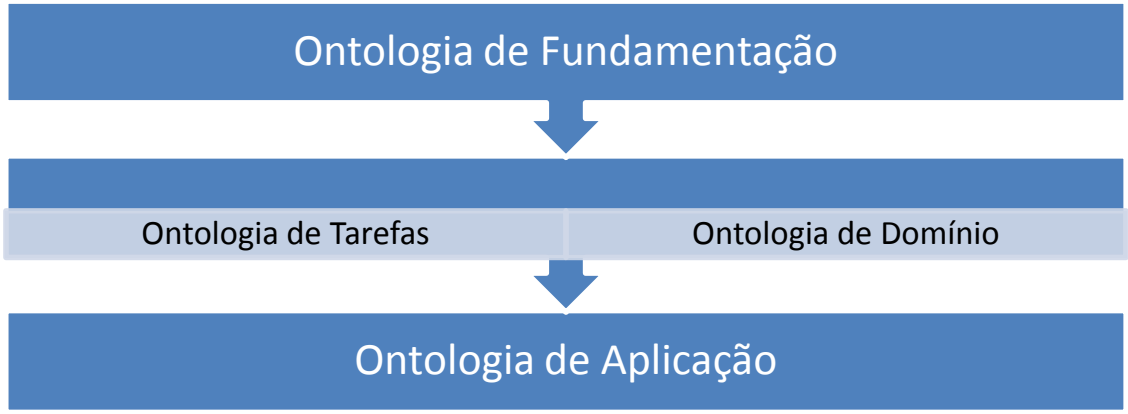

Figura 1 - Níveis Ontológicos

2.1.3. ONTOLOGIAS DE FUNDAMENTAÇÃO

Uma Ontologia de fundamentação define categorias ontológicas gerais, para domínios não específicos. As Ontologias de fundamentação são teorias axiomáticas independentes de domínio, seu objeto de estudo são conceitos como objeto, atributo, evento, mereologia, dependência e conexões espaços-temporais (SCHNEIDER, 2003).

Ontologias de fundamentação definem categorias ontológicas para domínios não específicos, a fim de criar um repositório de construtores primitivos para representação do conhecimento e formar uma base para a elaboração de ontologias de domínios específicos.

É possível fazer uma analogia com a Lógica Formal: assim como a Lógica Formal lida com a estrutura da lógica, independente da veracidade, as Ontologias de Fundamentação lidam com aspectos dos objetos, sem considerar as particularidades de cada um deles. Uma Ontologia de fundamentação, além de dever ser consistente com as teorias filosóficas e de cognição, deve ser precisa para expressar o conhecimento (GUIZZARDI, 2005).

Uma Ontologia cria uma abstração de conceitos gerais, que seja expressiva o bastante para expressar todos os conceitos possíveis, porém restringindo a criação de abstrações que não mapeiem para conceitos possíveis de serem instanciados.

A utilização destas teorias na construção da Ontologia geral é que produzem as características necessárias nesta para que as representações criadas no modelo correspondam a conceitos possíveis no domínio.

Como exemplo de Ontologias genéricas é possível citar a DOLCE(ALDO GANGEMI, MASOLO, *et al.*, 2002), GFO (HERRE, HELLER, *et al.*, 2006), UFO(GUIZZARDI e WAGNER, 2004) etc. Durante o desenvolvimento deste trabalho utilizaremos a UFO, pois dela deriva a linguagem ontologicamente bem fundamentada OntoUML também utilizada neste trabalho.

2.1.4. UFO – UNIFIED FOUNDATIONAL ONTOLOGY

A UFO é uma Ontologia construída com base em trabalhos de linguística, filosofia da linguagem, psicologia cognitiva e ontologias filosóficas. Além disso, diferentemente de outras Ontologias de Fundamentação, a UFO foi construída com o objetivo específico de servir como teoria com poder explanatório para resolver problemas clássicos de modelagem e interpretação em modelagem conceitual (GUIZZARDI e WAGNER, 2010). Desta maneira UFO se apresenta como uma alternativa adequada para servir de base para avaliação e (re)engenharia de linguagens de modelagem conceitual.

A UFO é divida em três camadas, sendo estes denominados UFO-A, UFO-B e UFO-C. Neste trabalho atem-se um fragmento específico da UFO-A, que trabalha com uma subclasse de tipos de objetos chamados Sortais Rígidos.

2.1.5. UNIVERSAIS E SUAS METAPROPRIEDADES

Universais e indivíduos são representados usualmente em linguagens conceituais como Tipos e instâncias. Primeiramente será discutido os Universais e as distinções ontológicas feitas pela UFO-A no que tange esta categoria de entidades. As distinções ontológicas existentes na UFO se materializam a partir de diferentes valores de metapropriedades dos Universais, portanto, para compreender as distinções entre os metatipos é necessário compreender estas metapropriedades. Além disso, as metapropriedades são relavantes para o entendimento deste trabalho, pois, as metapropriedades serão utilizadas para inferir o estereótipo de um determinado tipo.

Diferentemente da UML que identifica todos os universais com a metaclasse *Class* a UFO distingue entre os vários tipos de Universais. A Figura 2 demonstra a tipologia dos Universais da UFO-A.

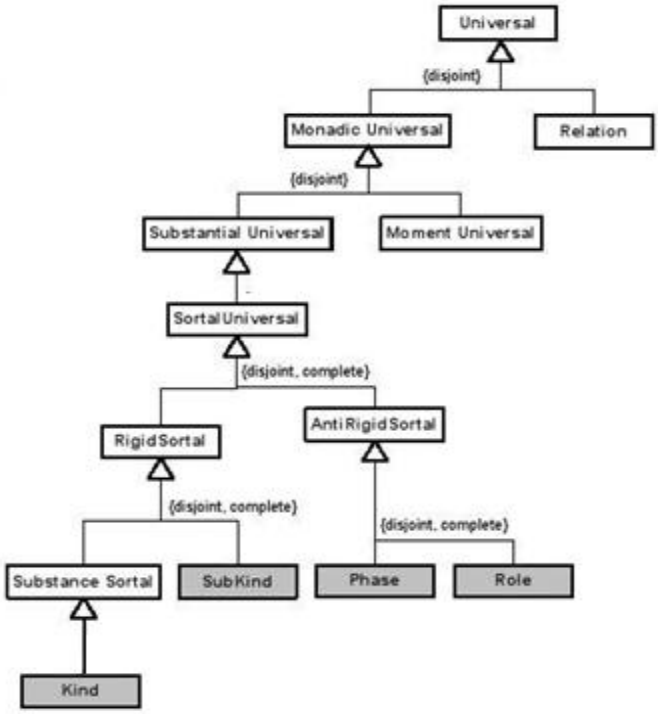

Figura 2 - Distinções Ontológicas de Substancias Universais

Substanciais são indivíduos existencialmente independentes, ou seja, sendo **x** um substancial, não existe outro indivíduo **y** que seja necessário para a existência do indivíduo **x**. Exemplos de substanciais são objetos como pessoa, uma casa, um martelo um carro, também os Objetos Fiat(SMITH, 2001), tal qual o estado do Espírito Santo e suas partes, distritos postais e áreas para não fumantes.

Por outro lado, *Moments* são indivíduos existencialmente dependentes, tal qual uma cor, uma conexão ou um pedido de compra. (BRAGA, ALMEIDA, *et al.*, 2010)

Sortais são tipos substanciais que proveem princípios de aplicação e de identidade para as suas instâncias. O princípio de aplicação concerne que nós estamos aptos a julgar se um tipo se aplica para um indivíduo particular, como exemplos de sortais pode-se listar: cachorro, pessoa, estudante, etc.; O princípio de aplicação traz como resultado prático

que na linguagem natural usualmente Sortais são os substantivos comuns(GUARINO, 1994).

Os tipos Sortal Antirrígido (*AntiRigidSortal*) e Sortal Rígido (*RigidSortal*) separa os tipos quanto ao seu princípio de rigidez. O conceito de **rigidez** aplicado a um sortal S define que para todas as instâncias *x* que instanciam S, será instância de S em $w \in W$, no qual *W* é o conjunto de todos os mundos possíveis. Um conceito pode ser rígido o que define que ele é aderente a esta definição do conceito de rigidez, logo, para todos y instância de um tipo **X**, y sempre será instância de X em todos os mundos possíveis, estes tipos são claramente instâncias de *RigidSortal*.

Um tipo não rígido pode ser semirrígido ou antirrígido. O valor semirrígido define um tipo mais fraco de não rigidez: existe *y* instância do tipo **X**, que dado dois mundos *w* e *w'*, *y* não seja mais instância de **X** em *w'*. Já o valor antirrígido define que dado um *x* instância do tipo **X**, obrigatoriamente existe um mundo ao qual a instância *x* não pertence ao tipo **X**. Os tipos antirrígidos são instâncias de *AntiRigidSortal*.

Neste fragmento da UFO-A os tipos rígidos são divididos em dois tipos quanto ao seu princípio de identidade. Enquanto os tipos Sortais Substanciais proveem o princípio de identidade, os tipos *SubKind* herdam este princípio de identidade de um supertipo.

O princípio de **identidade** provê suporte para o julgamento se dois indivíduos são o mesmo, por exemplo, identificar se o copo que você colocou sobre a mesa ontem é o mesmo copo que está sobre a mesa agora. Além disso, este princípio prevê suporte para distinguir quais mudanças podem ocorrer a um objeto sem que este deixe de ser o mesmo, ou seja, sem que ele mude de identidade. Em outras palavras, podemos dizer que um princípio de identidade torna possível decidir se dado duas instâncias **x** e **x'**, estas instâncias são as mesmas.

Por fim os tipos Antirrígidos podem ser tipos de Fase (*Phase*) ou tipos de Papéis *(Role).* Eles são diferenciados pelo princípio de **dependência**, sendo os tipos *Role* relacionalmente dependentes. O conceito de dependência define se a existência de uma entidade é dependente da existência ou da relação com outra entidade. A dependência pode ser rígida o que significa que a existência de um indivíduo implica necessariamente na existência de outro indivíduo específico. Em contraste, a dependência pode ser genérica quando a existência de um indivíduo necessariamente

implica na existência de um indivíduo pertencente a uma classe específica. Por fim, uma dependência relacional em nível de tipos é um tipo de dependência genérica que ocorre quando um indivíduo para instanciar uma classe X necessita da existência de um indivíduo da classe Y, de modo que estes indivíduos estarão relacionados através de uma relação R(GUIZZARDI, WAGNER, *et al.*, 2004).

Um exemplo de dependência rígida poderia ser aquele existente entre ser uma pessoa e o seu cérebro. Em contraste, a relação entre uma pessoa e o seu coração é um exemplo de uma dependência genérica. A relação entre o tipo estudante e o tipo escola é um exemplo de uma dependência relacional em nível de tipo, dado uma que uma pessoa é considerada um estudante se e somente se estiver matriculada em uma escola.

Essas são as principais metapropriedades utilizadas para fazer as distinções ontológicas presentes no fragmento da UFO-A abordado, logo os vários subtipos tem diferente valores para estas metapropriedades.

2.2. ONTOUML

A OntoUML é uma linguagem ontologicamente bem fundamentada. Esta linguagem é fruto da revisão da UML utilizando o arcabouço de revisão de linguagens da UFO proposto em (GUIZZARDI, 2005) e as distinções ontológicas presentes na UFO. Esta revisão demonstra que a UML sofre de incompletude ontológica, pois não é capaz de representar todas as distinções ontológicas feitas pela UFO (GUIZZARDI, 2005). Toda a Tipologia de Universais da UFO, e consequentemente as distinções feitas por cada construtor, é representado em UML unicamente pela metaclasse *Class*, logo a UML não distingue por exemplo entre tipos rígidos e antirrigidos ou entre fases e papeis.

A OntoUML é concebida como um fruto da revisão da UML considerando estas deficiências da linguagem. A UML provê dois tipos de mecanismos para modificar os elementos da linguagem: (i) especialização do metamodelo da UML, para adicionar novas semânticas para os elementos da UML; (ii) Modificar o modelo MOF para adicionar novos elementos ao metamodelo da UML. O primeiro mecanismo denominado *lightweigh extension* e o segundo *heavyweight extension*.(GUIZZARDI, 2005)

A OntoUML é proposta como uma *lightweigh extension* para o metamodelo da UML, especializando o metaconceito *Class* da UML. Esta especialização origina um perfil (*profile*) UML contendo um conjunto de classes estereotipadas. É possível visualizar um fragmento do metamodelo da OntoUML na Figura 3. Um estudo extensivo bem como uma implementação proposta do metamodelo da OntoUML podem ser encontrados em (BENEVIDES, 2007).

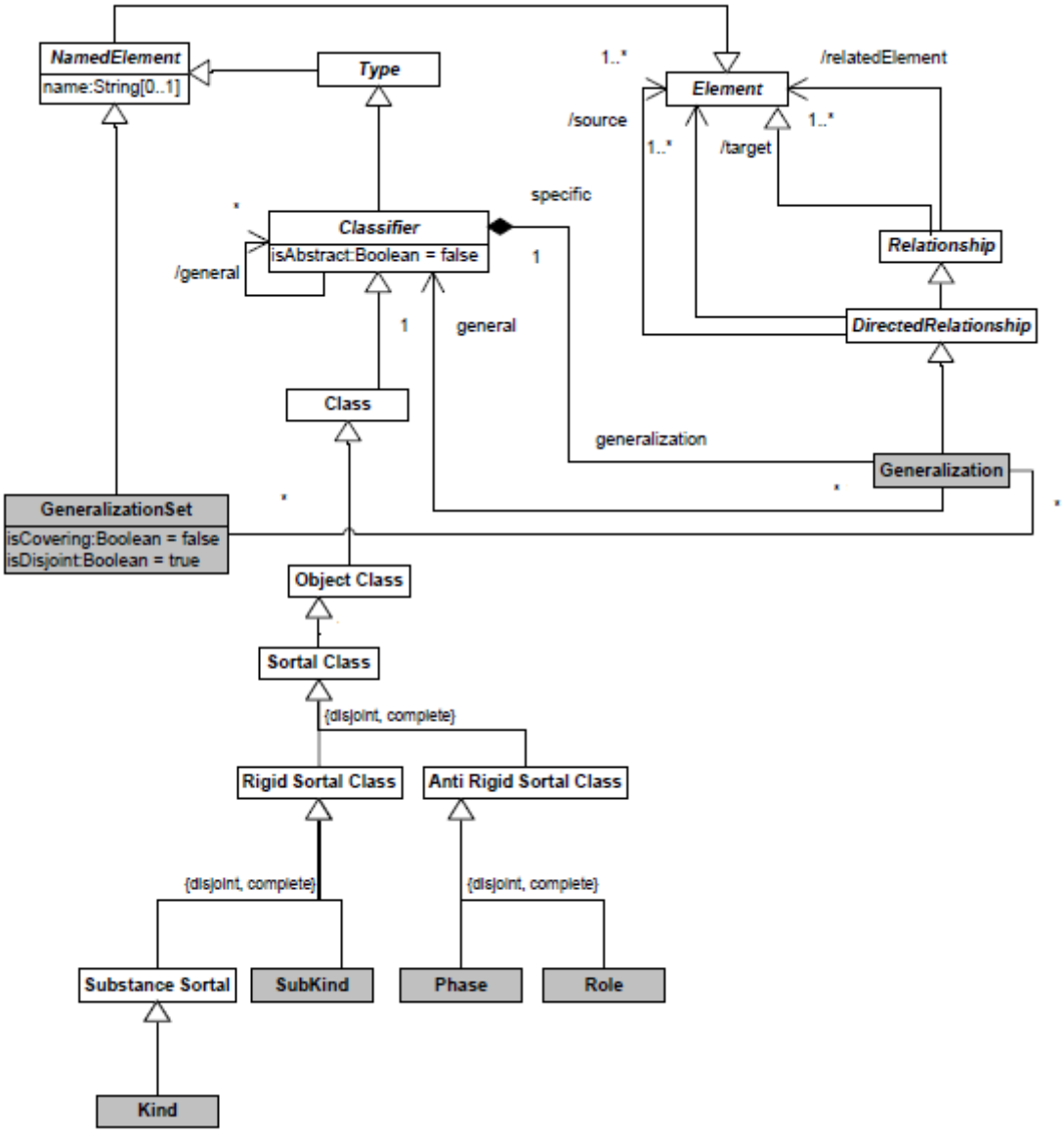

Figura 3 - Fragmento do Metamodelo da OntoUML

2.2.1. SORTAIS

Os construtores da OntoUML beneficiam-se das distinções ontológicas da UFO para aumentar a expressividade da linguagem. Além disso, o metamodelo da OntoUML contém restrições para evitar a criação de modelos que não sejam condizentes com a Ontologia de fundamentação UFO.

O estereótipo « *Kind* » representa um tipo sortal rígido, consequentemente, significa que suas instâncias proveem o princípio de identidade e aplicação, e o princípio de rigidez como abordado anteriormente. Um tipo estereotipado como « *Kind* » só pode ser subtipo de outro tipo rígido. Todo o objeto da OntoUML deve ser direta ou indiretamente um tipo estereotipado como « *Kind* » Além disso, nenhum tipo pode ser subtipo de mais de um tipo estereotipado como « *Kind* ».

Tipos estereotipados como « *Subkind* », são sortais rígidos, entretanto herdam o principio de identidade de um supertipo estereotipado como « *Kind* ». Um tipo estereotipado como « *Subkind* » só pode ser subtipo de outro tipo rígido. Exemplo de tipos estereotipados como Subkind pode ser visualisado

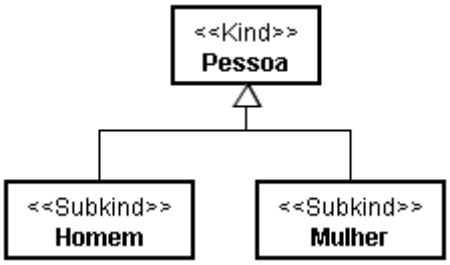

Figura 4 - Exemplo de tipos estereotipados como SubKind

O estereotipo « *Phase* » (Fase) representa um Sortal Antirrígido. Fases sempre ocorrem nas chamadas *Partições de Fase (Phase Partition)*. Um exemplo, representado na Figura 5, de uma partição de um sortal faseado seria, ⟨Vivo,Morto⟩, sendo Vivo e Morto respectivamente fases do sortal pessoa.

Toda a superclasse de uma partição de fases deve ser abstrata. Uma classe abstrata é representada em UML e consequentemente em OntoUML com o nome estilizado em itálico.

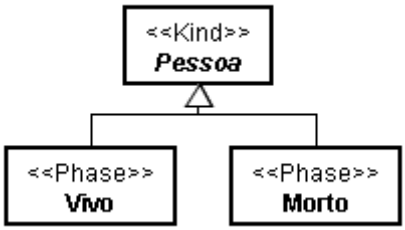

Figura 5 - Representação dos tipos de Fases de uma Pessoa

O estereotipo « *Role* » demonstra um papel que pode ser representado por instâncias de um sortal, sendo como o construtor « *Phase* », um sortal Antirrígido. Entretanto um role é relacionalmente dependente, ou seja, é necessária uma relação extrínseca com outra entidade para que uma instância Sortal possa passar a ser (ou deixar de ser) também instância de um « *Role* ». Enquanto isso a transição entre as fases do sortal no « *Phase* » se dá por uma característica intrínseca da entidade.

É possível visualizar um exemplo na Figura 6 onde o relacionamento é um axioma de restrição e de derivação. Um estudante deve estar matriculado na escola, e toda a pessoa que está matriculado em uma escola é um estudante.

Uma definição formal seria: **Seja X uma classe instância do construtor « role » e r uma associação que representa a condição de restrição de X. Logo a cardinalidade mínima de X.r deve ser ao menos 1 (#X.r ≥ 1).**(GUIZZARDI, WAGNER, *et al.*, 2004)

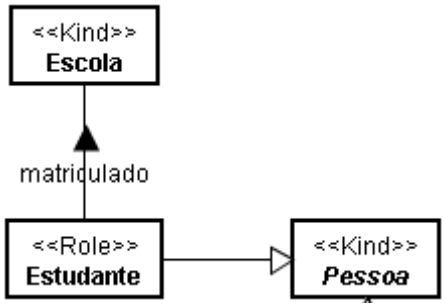

Figura 6 - Exemplo da Utilização do Construtor *Role*

Ambos os estereótipos « *role* » e « *phase* », representam tipos antirrígidos, logo tipos estereotipados por um destes construtores só podem ser supertipos de outros tipos estereotipados como « *role* » ou « *phase* ».

Tabela 1 - Resumo dos Sortais.

Estereótipo	Restrições
	Sortais Rígidos
Kind	Supertipos devem ser rígidos. Todo o conceito é direta ou indiretamente uma instância de um e apenas um tipo estereotipado como Kind.
SubKind	Supertipos devem ser rígidos.
	Sortais Antirrígidos
Phase	Sempre definido como uma partição. Para toda a Partição PS, feito sobre o Sortal S, esse deve ser abstrato. Consequentemente uma instância S deve, obrigatoriamente, ser instância de uma das fases da partição do Sortal.
Role	Seja X uma classe instância do construtor « role » e r uma associação que representa a condição de restrição de X. Logo , (#X.r ≥ 1).

2.2.2. ONTOLOGIAS DE DOMÍNIO OU DE TAREFA

Ontologias de domínio ou tarefa é o estudo do que existe em um domínio genérico como, por exemplo, automóveis ou medicina, ou alguma tarefa ou atividade como, por exemplo, gerência ou colaboração. Ontologias de domínio têm o objetivo de: compartilhar informações, interoperabilidade entre aplicações, reuso, etc.

Ontologias de domínio precisam ser representadas em uma linguagem. Em (GUIZZARDI, 2005), é defendido que uma linguagem de modelagem conceitual e representação ontológica de domínios devem ser baseadas em uma ontologia de fundamentação.

Ontologias de domínio construídas sobre uma linguagem baseada em uma ontologia de fundamentação garantem uma qualidade mínima, pois serão consistentes com as teorias apresentadas. As teorias presentes nas ontologias de fundamentação são utilizadas como um meio de avaliação da qualidade da ontologia de domínio (BLOMQVIST, ÖHGREN

e SANDKUHL, 2006). Como discutido na seção anterior, um exemplo de linguagem deste tipo é a OntoUML, cujo metamodelo é baseado na Ontologia genérica UFO.

Ainda segundo (Guizzardi, 2000) os conceitos e relações formam a base da ontologia. Entretanto, uma característica essencial de ontologias é a definição de axiomas. Simplesmente propor uma taxonomia ou um conjunto de termos básicos, não constitui uma ontologia. Os axiomas especificam definições de termos na ontologia e restrições sobre sua interpretação.

Os axiomas em uma ontologia podem apresentar formas e propósitos diferentes: Dois tipos de particular interesse são *axiomas de derivação* e os *axiomas de consolidação* (GUIZZARDI, FALBO e FILHO, 2002). Axiomas de consolidação têm o intuito de manter a coerência das informações existentes. Axiomas de derivação são aqueles que permitem explicitar informações a partir do conhecimento previamente existente. Assim, são meios para a dedução e representam consequências lógicas neste processo. Estes axiomas podem ser expressos em linguagem formal ou textual. Em OntoUML elas são preferencialmente expressas em Object Constraint Language(OCL) os axiomas de derivação também podem ser expressos em OCL ou visualmente através de uma extensão denominada OntoUML-R que dá suporte para regras de derivação(GRAÇAS, 2008).

2.2.3. ONTOLOGIAS DE APLICAÇÃO

De acordo com (GUARINO, 1997) uma ontologia de aplicação é uma especialização de uma ontologia geral, no caso ontologia de domínio e tarefas. Enquanto ontologias de tarefas e domínio têm definições gerais uma ontologia de aplicação contem definições específicas para uma classe de aplicações. Estas ontologias usualmente correspondem a papéis exercidos pelas entidades das ontologias de domínio quando executando uma atividade específica(GUIZZARDI, 2005). Um exemplo de ontologia de aplicação é a Ocean, uma ontologia de aplicação especialização de uma ontologia do domínio de colaboração(FRECHIANI, 2009).

2.3. TECNOLOGIAS PARA DESENVOLVIMENTO E REPRESENTAÇÃO COMPUTACIONAL DE METAMODELOS

Para a criação deste trabalho foi necessária á utilização de algumas tecnologias de metamodelagem. Estas tecnologias baseiam-se em MDE, Model-Driven Engineering, ou em tradução livre, Engenharia guiada por modelos. A iniciativa mais famosa em MDE é a MDA, do Object Management Group(OMG, 2003).

Várias destas tecnologias foram implementadas pela fundação Eclipse no seu ambiente de desenvolvimento, formando assim um arcabouço para criação de ferramentas baseadas em modelos.

2.3.1. MOF – META OBJECT FACILITY

O MOF(OMG, 2006), provê um arcabouço de gerenciamento de metadados e um conjunto de serviços para habilitar o desenvolvimento e interoperabilidade sintática de sistemas dirigidos a modelos e metadados.

Várias tecnologias padronizadas pela OMG, incluindo UML, XMI, e perfis (*profiles*) UML, utilizam-se do framework MOF para interoperabilidade sintática de metadados e manipulação de metadados.

Uma visão simplista do MOF seria vê-lo como uma linguagem para representação de metamodelos. Quando duas linguagens diferentes têm os seus metamodelos representados em MOF é possível a tradução entre estes modelos definidos em linguagens diferentes.

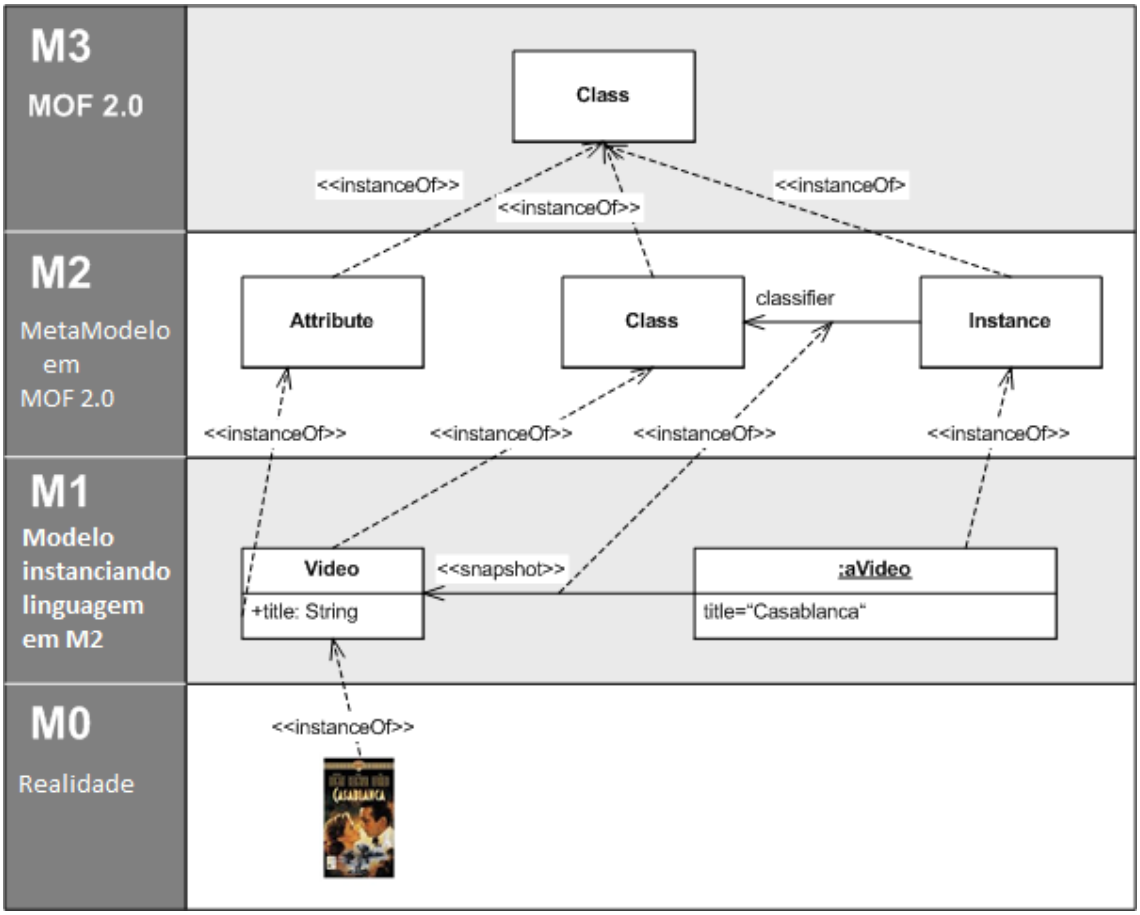

Figura 7 - Ilustração sobre a arquitetura em camadas do MOF.

Na Figura 7 é demonstrado uma ilustração do funcionamento da arquitetura MOF. O relacionamento estereotipado com « *instanceOf* » define que existe um relacionamento de instanciação entre as duas entidades relacionadas.

A camada mais alta M3 é o metametamodelo, o qual define o MOF. A camada M2 é uma instância desta camada, ou seja, o metamodelo das linguagens definidas nessa camada utilizam o MOF como linguagem. Como exemplo podemos citar as linguagens UML e a própria OntoUML. A camada M1 são os modelos propriamente que representam conceituações sobre um determinado domínio na realidade, estes modelos são criados utilizando as linguagens definidas em M2. A camada M0 é a constituída por instâncias de M1, por indivíduos de interesse em uma determinada porção da realidade.

Além do MOF, a OMG define o EMOF (Essential Meta-Object Facility)(OMG, 2006) , um subconjunto do *framework* MOF para a criação de metamodelos simples, no qual o ECore(STEINBERG, BUDINSKY, *et al.*, 2009), linguagem de definição de

metamodelos utilizada aqui é completamente baseado, modificando apenas a nomenclatura.

2.3.2. EMF – ECLIPSE MODELING FRAMEWORK

Para a criação da ferramenta proposta em (BENEVIDES, 2007) e também para a criação da ferramenta apresentada neste trabalho, foi utilizado o plug-in EMF(Eclipse Modeling Framework) do Eclipse(STEINBERG, BUDINSKY, *et al.*, 2009). Este plug-in é capaz de criar um editor em árvore para a construção de modelos a partir de um metamodelo construído em ECore.

O ECore é uma linguagem, praticamente igual ao EMOF, no qual o metamodelo da OntoUML foi codificado (MOORE, DEAN, *et al.*, 2004). Além do ECore, foi utilizada também para a construção do metamodelo OntoUML a linguagem OCL para representar as restrições existentes na axiomatização da ontologia UFO. Para a implementação do metamodelo OntoUML das restrições em OCL incorporadas aquele metamodelo, o leitor interessado deve consultar(BENEVIDES, 2007).

2.3.1. OCL – OBJECT CONSTRAINT LANGUAGE

Um diagrama MOF, ou expresso em UML, não é capaz de expressar todos os aspectos relevantes de um modelo. Existe, entre outras necessidades, a de descrever restrições adicionais sobre os objetos de um modelo. Frequentemente estas restrições são especificadas em linguagem natural, o que pode levar a ambiguidades. Para evitar estes problemas foi desenvolvida a linguagem OCL(Object Constraint Language)(OMG, 2006).

OCL é uma linguagem declarativa. Consequentemente uma expressão OCL garantidamente não contem efeitos colaterais, i.e. uma expressão OCL não alterará o valor das propriedades de um objeto. A execução de uma expressão em OCL, somente pode retornar um valor (OMG, 2006). Além disso, é possível executar consultas a um modelo utilizando a OCL, análogamente a linguagem SQL para banco de dados.

Neste trabalho, além das restrições OCL já presentes no metamodelo da OntoUML como proposto em (BENEVIDES, 2007), a linguagem foi utilizada para executar várias consultas sobre os modelos produzidos em OntoUML.

O arcabouço para interpretar as consultas OCL utilizado neste trabalho é o MDT (Model Development Tools)(ECLIPSE FOUNDATION, 2005), que foi escolhido devido à integração com a ferramenta de desenvolvimento eclipse utilizada neste trabalho e já ser construído para operar sobre modelos ECore.

2.3.1.1. SINTAXE DA LINGUAGEM OCL

Para a compreensão de algumas partes deste trabalho é necessário uma compreensão mínima da linguagem OCL. Desta maneira, a seguir são brevemente explicados e exemplificados alguns construtores da linguagem que são frequentemente referenciados neste trabalho.

O contexto de uma expressão OCL pode ser uma classe de objetos ou uma operação aplicável a um objeto. Para representar um contexto em OCL utilizamos a palavra reservada context.

A palavra chave *self* em OCL, serve para referenciar uma instância de contexto, análogo ao *this* nas linguagens orientadas a objeto, JAVA, C++ e C#.

Em OCL existem três tipos de coleções: Set, Bag e Sequence;

- *Set*: coleção que não contém elementos duplicados;
- *Bag*: coleção que pode conter elementos duplicados;
- *Sequence*: Coleção onde os elementos estão ordenados segundo algum critério. Na coleção *sequence* também pode aparecer elementos duplicados.

Algumas operações importantes sobre as coleções são:

- *Select*: operação sobre uma coleção que permite selecionar dentre os seus elementos os que têm uma determinada característica. Por exemplo:

  collection -> select (expressão booleana)

- *ForAll*: verifica se todos os elementos de uma coleção respeitam uma condição determinada. O valor de retorno da avaliação dessa expressão é um valor boleano.

  Collection -> forAll (v: Type | expressão booleana com v)

- *Exists*: verifica se existe algum elemento de uma coleção que respeita uma determinada condição. O valor de retorno dessa expressão é um valor booleano.

  Collection -> exists (v: Type | expressão booleana com v)

- *oclIsKindOf*: Operação que retorna verdadeiro se o objeto a qual a expressão foi aplicada pertence ao tipo especificado, ou alguns do seus subtipos.

  Object -> oclIsKindOf (type: OclType) : boolean

- *oclIsTypeOf*: operação que retorna verdadeiro se o objeto ao qual a expressão foi aplicada é do tipo especificado diretamente, ou seja, esta operação não considera a cadeia transitiva de subtipos daquele tipo em questão (ao contrário da operação anterior).

  Object -> oclIsTypeOf (type: OclType) : boolean

2.4. EDITOR GRÁFICO PARA CONSTRUÇÃO DE MODELOS EM ONTOUML COMO PLUG-IN DO ECLIPSE

Em (BENEVIDES, 2007), é apresentado um editor gráfico para a construção de ontologias na linguagem OntoUML. Este editor é construído como um plug-in do Eclipse, utilizando-se do EMF (Eclipse Modeling Framework).

Este editor implementa as categorias e axiomatizações, do metamodelo da UFO, além de restringir a criação de modelos ontologicamente inconsistentes, a complexidade das restrições ontológicas, são escondidas do usuário e são apresentadas caixas de diálogos com mensagens amigáveis para restringir a criação de modelos incorretos. Dessa maneira retira-se do usuário a responsabilidade checar se o modelo respeita as restrições ontológicas da UFO.

Para a construção desse editor o metamodelo da OntoUML foi implementado em ECore, e adicionada restrições em OCL.Um estudo do metamodelo completo em ECore, além das restrições em OCL, pode ser encontrado em (BENEVIDES, 2007). Este mesmo metamodelo, conjuntamente com as restrições em OCL, será utilizado neste trabalho.

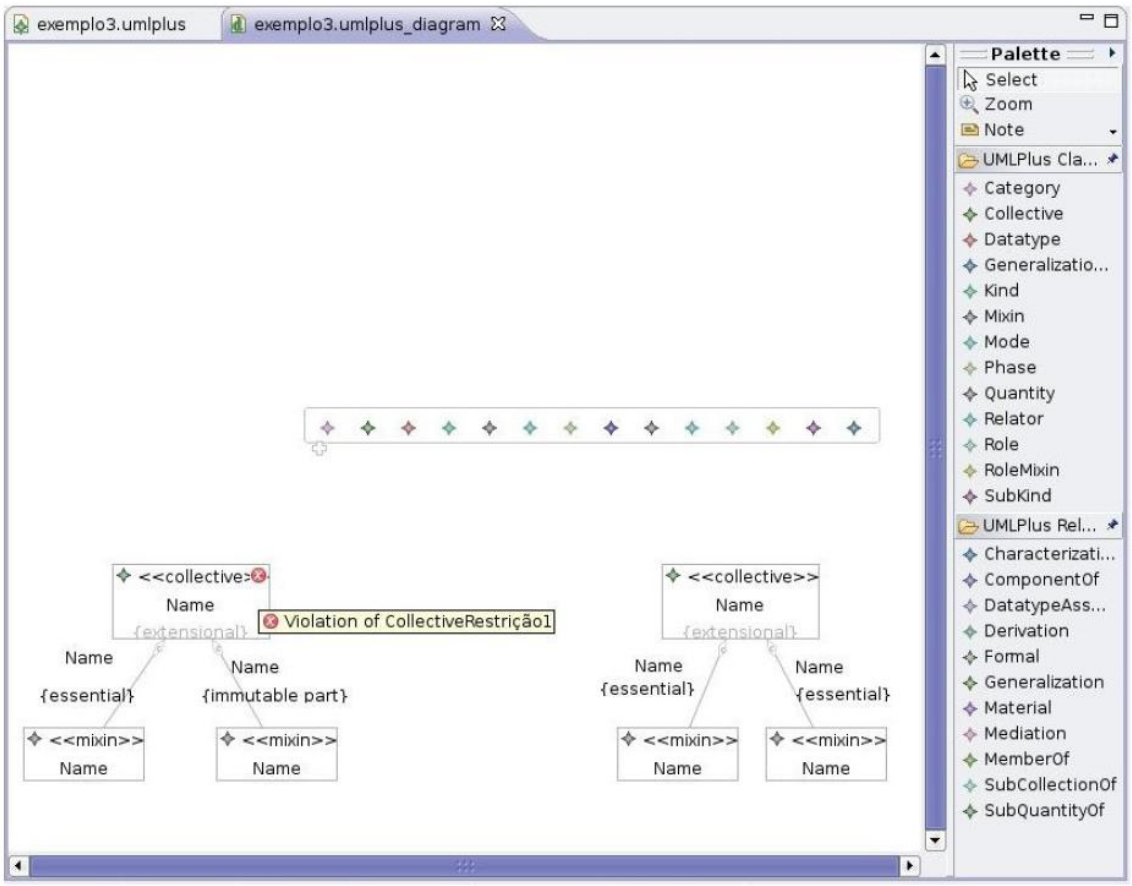

Figura 8 - Editor Grafico para OntoUML

Como é possível visualizar na Figura 8, o editor criado apresenta os diagramas com símbolos usualmente utilizados na UML, o que é natural visto que a OntoUML é uma é uma extensão dessa.

O Editor criado verifica os modelos construídos em dois momentos, no modo *live*, as restrições são checadas em tempo real, ou seja, sempre que e uma ação é executada sobre o modelo. Quando uma ação leva o modelo a ficar inconsistente imediatamente a ação é desfeita e uma caixa de diálogo com uma explicação para o usuário é apresentada.

Já no modo *batch* a verificação ocorre a pedido do usuário. Quando uma inconsistência no modelo é encontrada é criado um símbolo no construtor e uma mensagem explica a inconsistência. As correções têm que ser executadas manualmente pelo modelador. Algumas restrições são verificadas somente em modo *batch*.

2.5. DESIGN RATIONALE

Para poder compreender por que um projeto, ou modelo está da maneira que está é necessário compreender porque as escolhas feitas durante a construção foram as escolhas apropriadas. (MACLEAN, YOUNG e MORAN, 1989). Essas decisões que levam o projetista a chegar a certo estado de um projeto ou modelo são conhecidas como "*Design Rationale*".

Durante o processo de construção de um modelo, algumas vezes o artefato resultante não é o mais importante, mas sim as discussões e avaliações para obter um consenso acerca do domínio(MOORE e AGOGINO, 1987). Um dos pontos críticos nestes casos é entender as intenções e justificativas de várias decisões feitas por diferentes projetistas, as quais são coletivamente chamadas *Design Rationale*(MIZOGUCHI e IKEDA, 1998).

Design Rationale é uma explicação de porque o artefato, ou parte do artefato, está projetado da maneira que está (LEE e LAI, 1996). Além disso, pode incluir também informações sobre as alternativas disponíveis ao modelador e a argumentação que levou o projetista a decisão escolhida.

A falta da documentação do *Design Rationale* pode levar ao aumento da comunicação verbal necessária para o reuso(MEDEIROS e SCHWABE, 2006), ou até mesmo inviabilizar a reutilização (PRESUTTI e GANGEMI, 2008).

A pesquisa em *Design Rationale* normalmente envolve três aspectos principais: a captura, a representação e o uso. Geralmente, o esquema de representação determina os métodos usados para capturar e recuperar o *Design Rationale*.

A **captura** é o processo de adquirir o conhecimento acerca do *Rationale* da tarefa em execução. O *Design Rationale* são informações tediosas de serem registradas (capturadas) durante o desenvolvimento do projeto e difíceis de recuperar depois do fim da fase de projeto (GRUBER, BAUDIN, *et al.*, 1991). Além disso, no caso da construção de modelos, muitas vezes este *Rationale* não esta descrito em nenhum lugar, ou então está espalhado em coleção de documentos ou rascunhos(KLEIN, 1993). Logo um ponto crucial para a utilização do Design Rationale é a dificuldade da captura do *Design Rationale*, pois este processo usualmente requer que o projetista explique todas as suas ações o que aumenta muito carga de trabalho do projetista(MEDEIROS, 2006).

Após o processo de captura é necessário **representar** o *Design Rationale*, utilizando uma linguagem que alie expressividade com capacidade de processamento computacional. Entre várias metodologias de captura e representação de Design *Rationale* pode-se enumerar: IBIS (KUNZ e RITTEL, 1970); Liskov and Guttags(LISKOV e GUTTAG, 1986); Potts and Bruns(POTTS e BRUNS, 1988); SEURAT(BROWN, 2003);Kuaba(MEDEIROS, 2006).

A finalidade de todas as pesquisas feitas sobre *Design Rationale* é que estas informações sejam **usadas** para facilitar a colaboração entre vários modeladores durante a construção de um modelo e o entendimento do modelo. Neste trabalho o *design rationale* será utilizado para que o modelador compreenda as decisões de outros modeladores quando confrontados com decisões já tomadas por outros modeladores. Por exemplo, ao necessitar estereotipar um determinado tipo o modelador pode ter acesso aos motivos que levaram outro modelador a utilizar um determinado estereótipo quando confrontado com o mesmo questionamento.

Além disso, quando consideramos o caráter altamente colaborativo da construção de uma ontologia, dado que esta é vista como um consenso acerca de um domínio entre um grupo de pessoas(GUIZZARDI, 2005), entende-se que o processo de captura do *Design Rationale* é extremamente importante, pois além de armazenar o conhecimento adquirido durante o processo de construção da ontologia, este propicia um ambiente de colaboração e discussão acerca das decisões no processo de construção da ontologia.

2.6.CONCLUSÃO

Neste capítulo foram apresentados os pressupostos teóricos para o desenvolvimento deste trabalho. Primeiramente foi apresentada uma discussão sobre Ontologia, onde demonstramos os vários níveis ontológicos e apresentamos a Ontologia de fundamentação UFO e a linguagem construída OntoUML. A linguagem OntoUML é uma linguagem ontologicamente bem fundamentada, na qual modelos criados utilizados esta linguagem são consistentes com a Ontologia UFO.

Também foram apresentadas algumas tecnologias necessárias para o desenvolvimento deste trabalho. Várias destas tecnologias foram desenvolvidas pelo Object Management Group (OMG), entre elas, MOF, EMOF, OCL. O arcabouço de metamodelagem EMF,

desenvolvida pela fundação Eclipse, também foi amplamente utilizada no desenvolvimento deste trabalho.

Por final, foi apresentada algumas definições de Design Rationale. Esse será utilizado durante o desenvolvimento do trabalho para: (*i*) tornar as decisões tomadas anteriormente rastreáveis; (*ii*) facilitar a reuso das ontologias;.

CAPÍTULO 3. GUIA METODOLÓGICO PARA A CONSTRUÇÃO DE MODELOS UTILIZANDO A ONTOUML

Neste capítulo, é apresentado um guia metodológico para a construção de modelos utilizando a OntoUML. Primeiramente, é demonstrado que a OntoUML é uma linguagem de padrões e, depois, como é possível desenvolver regras de formação para estes padrões. Por último, é apresentado o DPModel, um modelo para armazenar informações relativas aos padrões de projetos presentes em um modelo.

A Engenharia de Ontologias compreende o conjunto de atividades feitas durante a conceitualização, projeto, implementação e publicação de ontologias (DEVEDZIC, 2002) e também determina como projetar gerenciar e explorar ontologias dentro de sistemas de informações (GANGEMI, 2007). Não existe uma única maneira ou metodologia correta de desenvolver ontologias, a melhor solução depende da aplicação desejada e das possíveis extensões que serão feitas na ontologia (MCGUINNESS, 2001). Como exemplo de metodologias para a construção de ontologias pode-se citar: Methontology(FERNÁNDEZ, GÓMES-PÉREZ e JUSTO, 1997), CO4 (EUZENAT, 1996), Uschold and King's(USCHOLD e KING, 1995).

Como discutido em (GUIZZARDI e HALPIN, 2008), tem-se reconhecido, cada vez mais, a necessidade de se ter linguagens e metodologias para construção de ontologias que incorporem distinções verdadeiramente ontológicas e diretrizes metodológicas expresivas.

A linguagem OntoUML tem sido utilizada com sucesso na criação de ontologias em diversos domínios complexos, como eletrofisiologia do coração (GONÇALVES, ZAMBORLINI e GUIZZARDI, 2009) e Petróleo e Gás (GUIZZARDI, LOPES, *et al.*, 2010). Além disso, tanto a linguagem quanto seus conceitos subjacentes têm experimentado uma crescente adoção (HALPIN e MORGAN, 2008)(BAUMAN, 2009).

Entretanto, tal linguagem tem sido considerada altamente complexa para modeladores iniciantes, pois requer do modelador, além do conhecimento acerca do domínio. o conhecimento acerca das distinções ontológicas e restrições sintáticas da linguagem. Logo, o objetivo neste capítulo é demonstrar que a OntoUML é uma linguagem de padrões, e que estes padrões podem ser utilizados para gerar regras de formação do

modelo. Desta maneira procura-se, aqui, prestar auxilio a modeladores inexperientes. Também é apresentado um modelo que visa extrair informações do modelo em relação aos padrões de projetos utilizados neste (GRAÇAS e GUIZZARDI, 2010). Este modelo é denominado DPModel, sendo DP uma abreviação de Padrões de Projetos no idioma inglês.

3.1. ONTOUML COMO UMA LINGUAGEM DE PADRÕES

Os padrões de projeto[3] têm o objetivo de identificar estruturas comuns de projetos para criar padrões reutilizáveis. Esses padrões proveem uma solução para problemas recorrentes. Usualmente, um padrão de projeto descreve quando aplicá-lo e as consequências da sua utilização(GAMMA, HELM, *et al.*, 1995).

Um padrão de projeto já compreende o *Design Rationale* empregado para a utilização deste. Consequentemente, além de facilitar a modelagem, padrões de projetos facilitam o entendimento da ontologia por outros modeladores. Os padrões de projetos podem emergir das melhores práticas detectadas por um modelador ou extraídas diretamente de ontologias de referência(PRESUTTI e GANGEMI, 2008).

Como será demonstrado neste capítulo, ao utilizarmos a linguagem OntoUML para a criação de ontologias, podemos facilmente identificar padrões de modelagem. A presença de um tipo com um estereótipo OntoUML, requer a presença de um bloco de modelagem, para que as regras de restrição da linguagem sejam satisfeitas.

É importante salientar que se definiu, neste trabalho, uma linguagem de padrões como uma linguagem na qual os construtores não são utilizados de maneira avulsa, mas sim dentro de um contexto de um padrão de projetos.

Para demonstrarmos essa propriedade da OntoUML demonstraremos que, no fragmento utilizado neste trabalho, cada construtor da OntoUML com exceção do estereótipo « *Kind* » ao serem utilizados requerem a presença de outros construtores, devido as restrições sintáticas da linguagem OntoUML.

---

[3] O termo padrão de projeto não faz referência a fases de desenvolvimento: análise, projeto etc. Eles são uteis tanto para projetar modelos independentemente da fase em que se encontra o projeto. O termo foi preservado para manter compatibilidade com a literatura.

Em cada padrão de projeto serão catalogados os papéis possíveis de serem instanciados por um elemento do domínio em questão. O papel de uma instância em um padrão de projeto define a função de um conceito em uma instância de um padrão de projeto.Esta informação será posteriormente utilizada para inferir o *design rationale* de uma instância e criar uma base histórica para auxiliar o modelador durante o processo de modelagem.

Os padrões de projetos que emergem da ontologia de referência UFO espelham as teorias estudadas durante a formação desta – teorias gerais independentes de domínio – logo os padrões que emergem da UFO também são padrões de projetos independente de domínio. Consequentemente estes padrões podem se manifestar em outras linguagens além da OntoUML.

3.1.1.PADRÃO SUBKIND

Tipos estereotipados como *Subkinds* são tipos rígidos e que não proveem princípios de identidade para suas instâncias. Logo, podem se manifestar de duas formas: como uma especialização de um tipo S, ou como parte de um conjunto de generalização que tem como superclasse comum um tipo S, sendo S um tipo estereotipado como Sortal Rígido ⟨Kind, Subkind⟩. Partições são representadas, em UML, e consequentemente OntoUML, pelo que é chamado de um Conjunto de Generalizações (*Generalization Set*) definido com os meta-atributos Disjunto e Completo (*Disjoint*, *Complete*)(GUIZZARDI, 2005). Na Figura 4, os *subkinds* ⟨Homem, Mulher⟩ definem uma partição do tipo Pessoa. As duas tabelas a seguir definem: (i) os papéis existentes em um padrão de projeto e quais os estereótipos possíveis para um papel (ii) as regras de formação do guia metodológico.

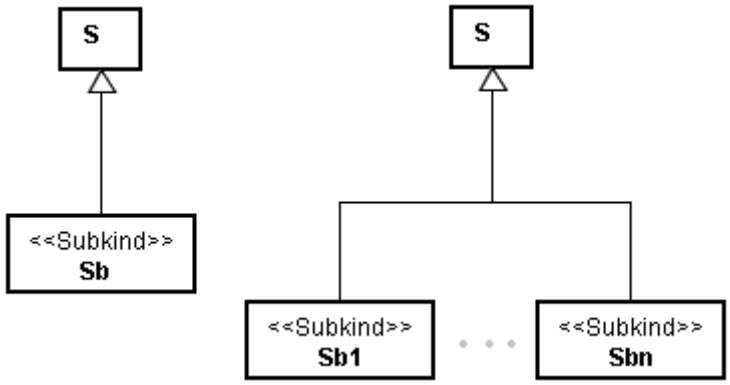

Figura 9 - Padrão Subkind

Tabela 2 - Tabela de Papeis no padrão Subkind

Papéis	Definição	Estereotipos Possíveis
SubKindSuperTypeDPR[4]	Qualquer tipo que seja supertipo de um ou mais tipos estereotipados como *SubKind*.	⟨Kind, Subkind⟩
SubkindDPR	Qualquer tipo estereotipado como *SubKind.*	⟨Subkind⟩

R_{SK} (X)	Regras Mutuamente Exclusivas	ii. Caso já exista o *generalization set* GS, X deve ser conectado como subclasse a GS, e os componentes de GS são todos *subkinds*; iii. Caso já exista S | S ∈ {*kind*, *subkind*}, X deve ser conectado ao tipo S existente. Se existir outro *subkind* SK conectado ou que deve ser criado e conectado como subclasse de S então é criado um *generalization set* GS tal que X e SK fazem parte de GS, e S é conectado a GS como supertipo comum de seus componentes; iv. Caso não exista S, deve ser criado um novo tipo S ∈ {*kind*, *subkind*}, S deverá ser conectada a X como superclasse, e caso S seja um *subkind*, a regra R_{SK} é aplicada a S. Se existir outro *subkind* SK que deve ser criado e conectado como subclasse de S então é criado um *generalization set* GS, tal que X e SK fazem parte de GS, e S é conectado a GS como supertipo comum de seus componentes;

[4] DPR: *Design Pattern Role* - Esta sigla foi adicionada ao final de cada nome para facilitar o entendimento que o termo se refere ao papel executado pela entidade no padrão de projeto

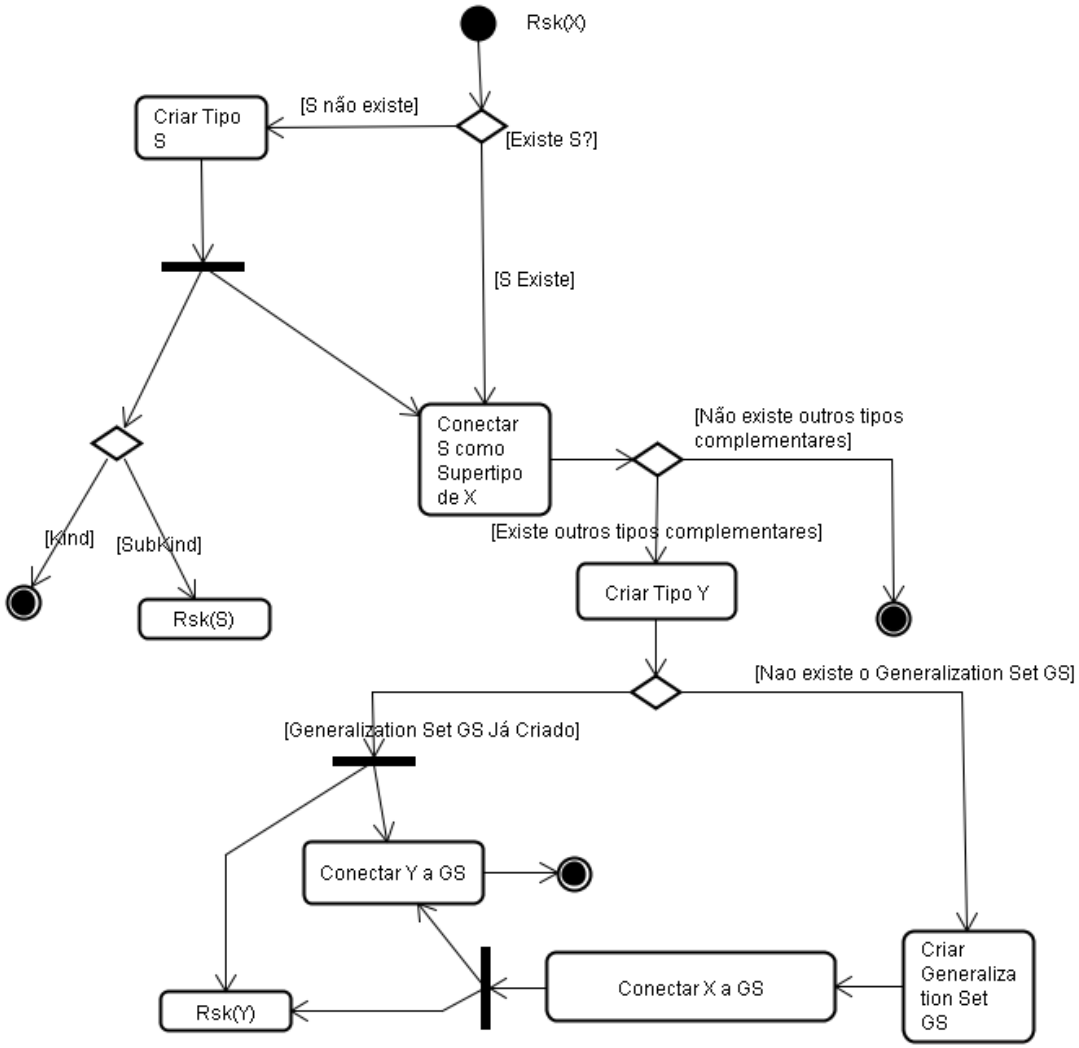

Figura 10 - Diagrama de Atividades do padrão SubKind

3.1.2. PADRÃO PHASE PARTITION

Phases se manifestam sempre como parte de uma *Phase Partition*, cuja superclasse comum é necessariamente um *Sortal* S. Este padrão é demonstrado na Figura 11. Dessa forma, pode-se criar uma regra de formação de modelos tal que toda vez que uma *Phase* P for instanciada – uma classe em OntoUML for estereotipada como *Phase* – a regra R_P descrita abaixo se aplica. Os estereótipos possíveis para cada papel e as regras de formação estão definidas abaixo.

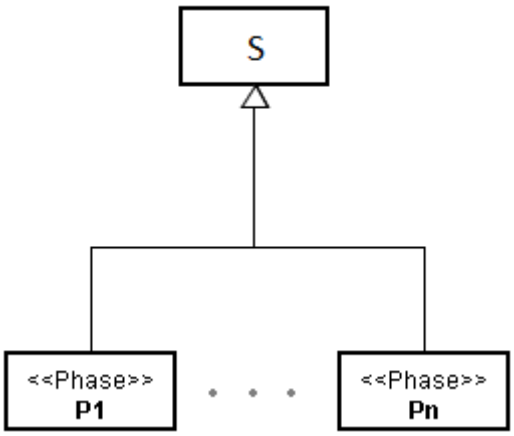

Figura 11 - Phase Partition

Tabela 3 - Tabela de papéis no Padrão Phase Partition

Papéis	Definição	Estereotipos Possíveis
PhasedSortalDPR	Qualquer tipo que seja supertipo de um ou mais tipos estereotipados como *Phase*.	⟨*Kind, Subkind,Role,Phase*⟩
PhaseDPR	Qualquer tipo estereotipado como *Phase*	⟨*Phase*⟩

<table>
<tr><td rowspan="5">R_P (X)</td><td rowspan="5">Regras Mutuamente Exclusivas</td><td colspan="2">i. Caso já exista PP, uma Phase Partition disjunta e completa, X deve ser conectado como subclasse;</td></tr>
<tr><td colspan="2">ii. Deve ser criada uma nova Phase Partition PP disjunta e completa, X deve ser conectado como subclasse a PP.</td></tr>
<tr><td rowspan="2">Regras Mutuamente Exclusivas</td><td>a.1. Caso já exista S | S ∈ {kind, subkind, phase, role}, PP deve ser conectado a S.</td></tr>
<tr><td>a.2. Caso não exista S | S ∈ {kind, subkind, phase, role} deverá ser criado, S deverá ser conectado a PP como superclasse, e uma das seguintes regras deverá ser aplicada a S: R_{SK} (seção 3.1.1), R_P, R_R (seção 3.1.3), caso S seja um Subkind, uma Phase, ou um Role, respectivamente.</td></tr>
<tr><td></td><td>b. Devem ser criadas novas phases $P_i..P_n$ complementares à X tal que cada uma dessas phases deve ser conectada a PP como subclasse;</td></tr>
</table>

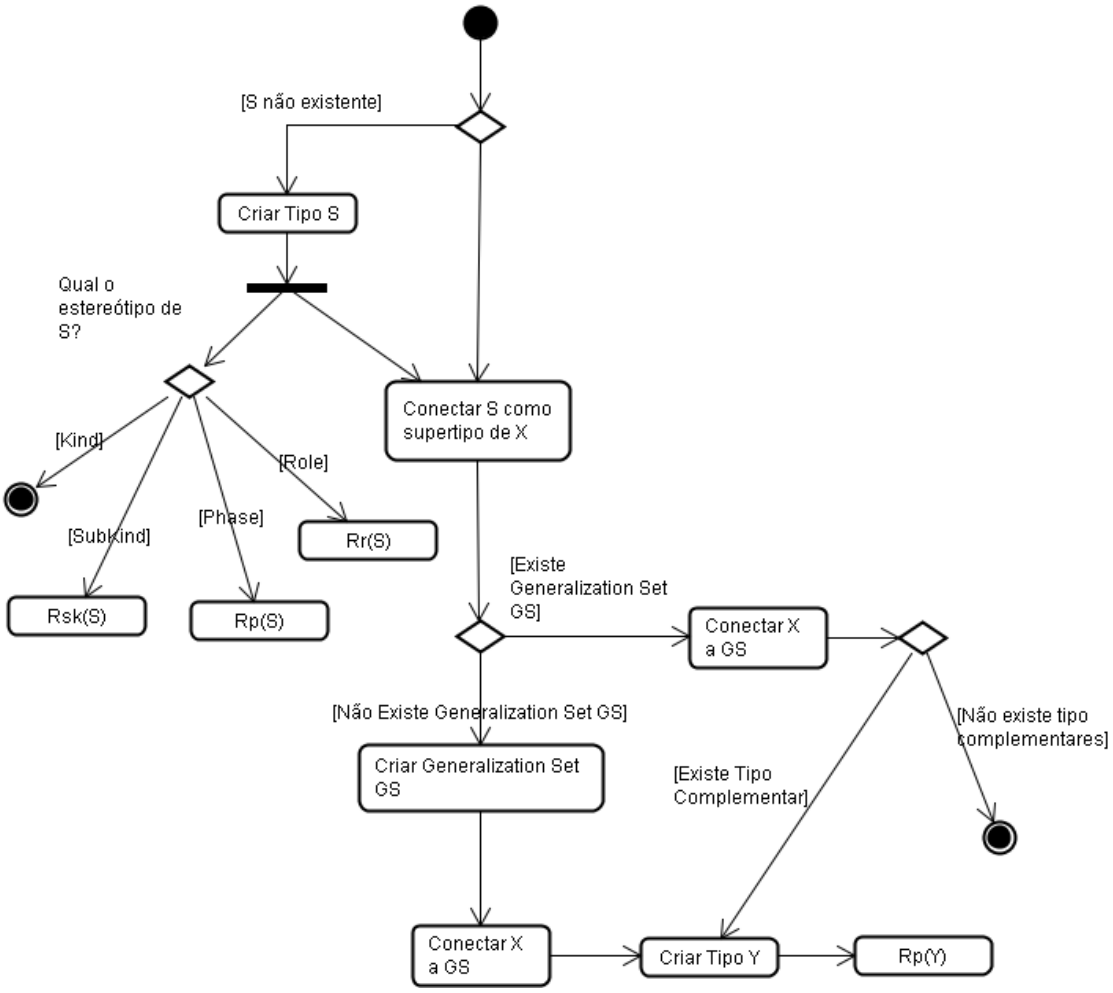

Figura 12 - Diagrama de Atividades do Padrão Phase Partition

3.1.3. PADRÃO ROLE

Assim como *Phases*, *Roles* representam especializações (possivelmente sucessivas) de um *Kind*, cuja condição de restrição é uma propriedade relacional. Desta forma, *Roles* se manifestam seguindo o padrão demonstrado na Figura 13. A regra de formação correspondente ao tipo *Role* é definida abaixo. A propriedade relacional deve contar a cardinalidade mínima igual a 1 para o classificador alvo da relação de dependência. Os papéis do padrão e os estereótipos possíveis, além da regra de formação são definidos em seguida.

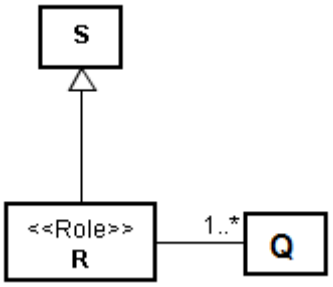

Figura 13 - Padrão Role. Sendo R um Role, C um Classificador qualquer e S um Sortal.

Tabela 4 - Tabela de papéis no padrão Role

Papéis	Definição	Estereotipos Possíveis
RoleSuperTypeDPR	Qualquer tipo que seja supertipo de um ou mais tipos estereotipados como *Role*.	⟨*Kind, Subkind,Phase,Role*⟩
RoleDPR	Qualquer tipo estereotipado como *Role*	⟨*Role*⟩
RoleTargetDPR	Qualquer tipo o qual um Role seja relacionalmente dependente.	⟨*Kind, Subkind,Phase,Role*⟩
RoleAssociationDPR	Relacionamento que expressa a dependência relacional do tipo estereotipado como *Role.*	-

<table>
<tr><td rowspan="4">R_R (X)</td><td rowspan="4"></td><td rowspan="2">Regras Mutuamente Exclusivas</td><td>i.a. Caso já exista o Tipo S | S ∈ {kind, subkind, phase, role}, X deve ser conectado a S, criando o relacionamento de generalização.</td></tr>
<tr><td>i.b. Caso não exista o tipo S | S ∈ {kind, subkind, phase, role}, S deverá ser criado conectado a X como supertipo, e uma das seguintes regras deverá ser aplicada a S: R_{SK} (seção 3.3), R_P (seção 3.1), R_R, caso S seja um Subkind, uma Phase, ou um Role, respectivamente.</td></tr>
<tr><td colspan="2">ii. Caso X já não esteja associado a um outro tipo T através de uma associação R e com cardinalidade mínima ≥ 1 na terminação ligada a T, então R deverá ser criada; a associação R deverá ser conectada em uma terminação (association end) ao tipo X e em outra terminação ao tipo Q (tal que Q ≠ X). A terminação associada ao tipo Q deve ter cardinalidade mínima ≥ 1;</td></tr>
<tr><td colspan="2">iii. O tipo Q pode ser um tipo existente ou deve ser criado. No último caso, uma das seguintes regras deverá ser aplicada a Q: R_{SK} (seção 2.3), R_P, R_R (seção 2.2.), caso Q seja um Subkind, uma Phase, ou um Role, respectivamente.</td></tr>
</table>

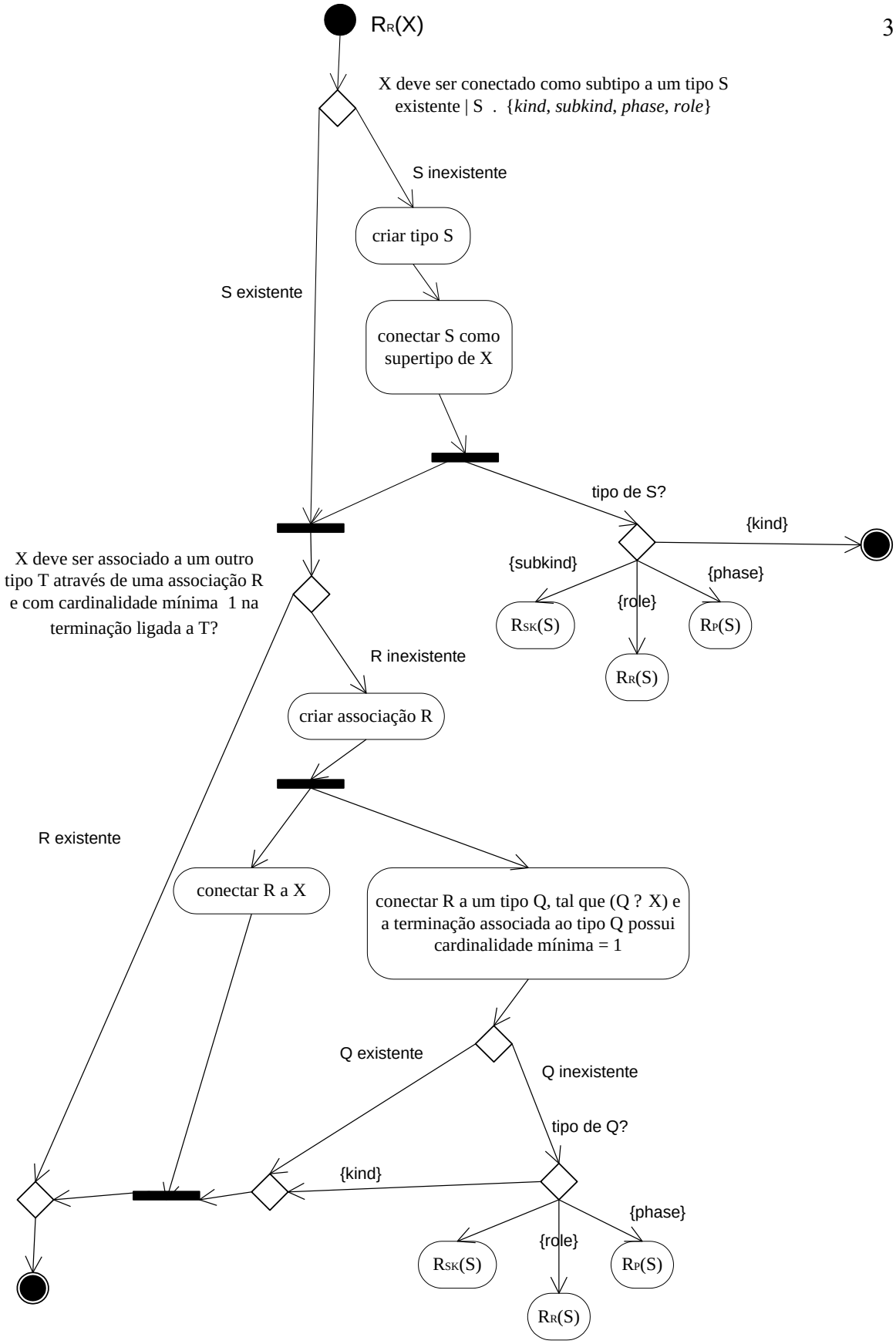

Figura 14 - Diagrama de Atividades do Padrão Role

3.2. ILUSTRAÇÃO DE UM GUIA METODOLÓGICO BASEADO NAS REGRAS DE FORMAÇÃO PROPOSTAS

O objetivo dessa seção é exemplificar a aplicação das regras de formação de modelos OntoUML propostas neste capítulo, usando como ilustração o modelo da Figura 15. A execução dessas regras é manifestada através de diálogos com o modelador, servindo como guia metodológico de apoio ao usuário na construção do modelo. É importante ressaltar que as regras de formação dos padrões da linguagem propostas nas seções anteriores foram projetadas para permitir que a definição do guia metodológico proposto a seguir pudesse ser feito de maneira completamente sistemática. Em outras palavras, o diálogo apresentado a seguir pode ser gerado automaticamente através da execução das regras de formação. Estes diálogos foram implementados na ferramenta apresentada no capítulo 5. Por fim, como será observado a seguir, através da execução desses diálogos entre modelador e ferramenta abstrata, o modelo da Figura 15 é construído de forma completamente sistemática. Os modelos resultantes da aplicação de cada um dos passos descritos a seguir na construção desse modelo são ilustrados na **Erro! Fonte de referência não encontrada.**.

Figura 15 - Exemplo de Modelo Conceitual em OntoUML

1. Suponha que o primeiro tipo incluído pelo usuário no modelo é o tipo *Criança*, e que este tipo é estereotipado como *Phase*.
2. A regra R_P *(Criança)* é então executada. Como não existe qualquer outro elemento do modelo, então o item (ii-b) é executado: *Criança é um subtipo de que? Resposta: Pessoa.* Em seguida, o item (ii-c) é executado: *Quais seriam as outras fases de Pessoa complementares a Criança? R: Adolescente, Adulto.* Por fim: *Existe um tipo T do qual Pessoa é um subtipo? R: Não.* Como conseqüência, nenhuma regra adicional é aplicada sobre *Pessoa*.

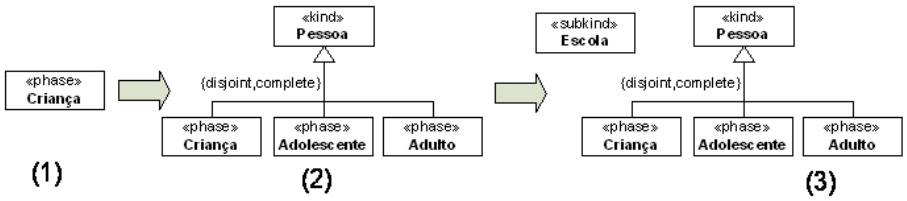

Figura 16 - Passos de formação de um modelo OntoUML pela aplicação das Regras de Construção propostas.

3. Suponha que o próximo tipo incluído seja *Escola*, e que este tipo é estereotipado como *subkind*.
4. A regra R_{SK} *(Escola)* é então executada. Como não existe uma partição formada por *subkinds* no modelo, as possibilidades de execução são os itens (ii) e (iii). Item (ii) é eliminado pela seguinte pergunta: *Escola é um tipo de Pessoa? R: Não.* Item (iii) é então executado: *Então, Escola é um subtipo de que? R: Empresa. Existe algum outro subtipo de Empresa e complementar a Escola que deve ser incluído no modelo? R: Não. Existe um tipo T do qual Empresa é um subtipo? R: Não.*
5. Suponha que o próximo tipo incluído seja *Estudante*, e que este tipo seja estereotipado como *Role*.

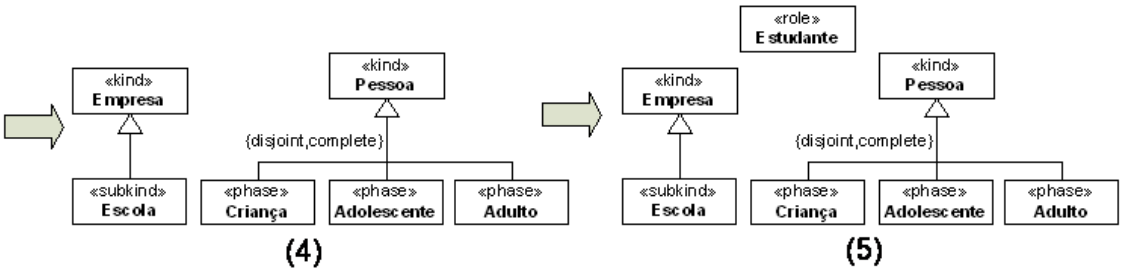

Figura 17 - Passos de formação de um modelo OntoUML pela aplicação das Regras de Construção propostas.

6. A regra R_R *(Estudante)* é então executada. As possibilidades de execução são os itens (i) e (ii). Item (i) é confirmado pela seguinte pergunta: *Estudante é um tipo de Empresa, Escola, Pessoa, Criança, Adolescente, ou Adulto? R: Pessoa.* Item (iii) é então executado: *Estudante é um papel que Pessoa desempenha quando relacionada com o que? Escola, Empresa, Criança, Adolescente ou Adulto? R: Escola. Como se chamaria esta relação? R: estuda em. Um Estudante estuda em no mínimo quantas Escolas? R: 1. Existe um máximo? R: Não. Uma Escola está relacionada com no mínimo quantos Estudantes? R: 1. Existe um máximo? R: Não.*

7. Suponha que o próximo tipo incluído seja *Mulher*, e que este tipo seja estereotipado como *subKind*.

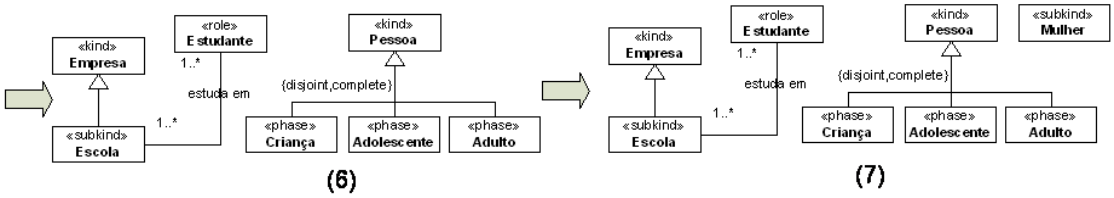

Figura 18 - Passos de formação de um modelo OntoUML pela aplicação das Regras de Construção propostas.

8. A regra R_{SK} *(Mulher)* é então executada. Existe neste momento somente um único *generalization set* formado somente por *subkinds*. O item (i) é então descartado pela seguinte pergunta: *Mulher é um subtipo de Empresa complementar a Escola? R: Não.* As possibilidades de execução são os itens (ii) e (iii). Item (ii) é confirmado pelas seguintes perguntas: *Mulher é um subtipo de Pessoa? R: Sim. Existe algum subtipo adicional de Pessoa complementar a Mulher que deve ser representado? R: Homem.*
9. Suponha que o próximo tipo incluído seja *Marido*, e que este tipo seja estereotipado como *Role*.

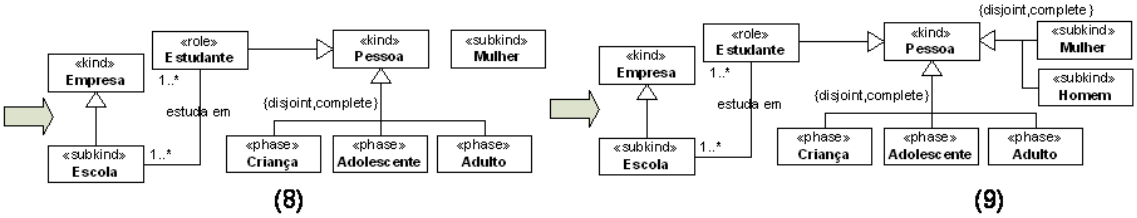

Figura 19 - Passos de formação de um modelo OntoUML pela aplicação das Regras de Construção propostas.

10. A regra R_R *(Marido)* é então executada. As possibilidades de execução são os itens (i) e (ii). Item (i) é confirmado pela seguinte pergunta: *Marido é um tipo de Empresa, Escola, Pessoa, Homem, Mulher, Criança, Adolescente, ou Adulto? R: Homem.* Item (iii) é então executado: *Marido é um papel que Homem desempenha quando relacionado com o que? Empresa, Escola, Pessoa, Mulher, Criança, Adolescente, Adulto? R: Esposa.*

    Como se chamaria esta relação? R: casado com. Um Marido é casado com no mínimo quantas Esposas?R: 1. Existe um máximo?R: 1. Uma Esposa está relacionada com no mínimo quantos Maridos?R: 1. Existe um máximo? R: 1.

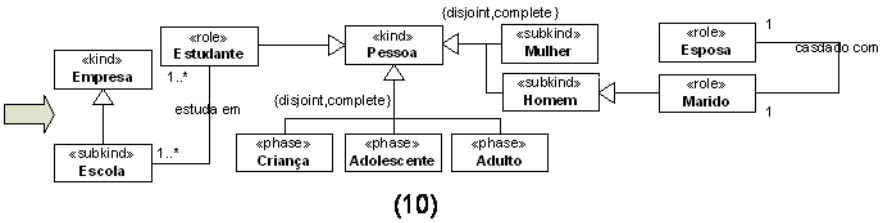

Figura 20 - Passos de formação de um modelo OntoUML pela aplicação das Regras de Construção propostas.

11. A regra R_R *(Esposa)* é então executada. As possibilidades de execução são os itens (i) e (ii). Item (i) é confirmado pela seguinte pergunta: *Esposa é um tipo de Empresa, Escola, Pessoa, Homem, Mulher, Criança, Adolescente, Adulto ou Marido? R: Mulher*. Como *Esposa* já está conectada a *Marido*, o item (iii) não é executado.

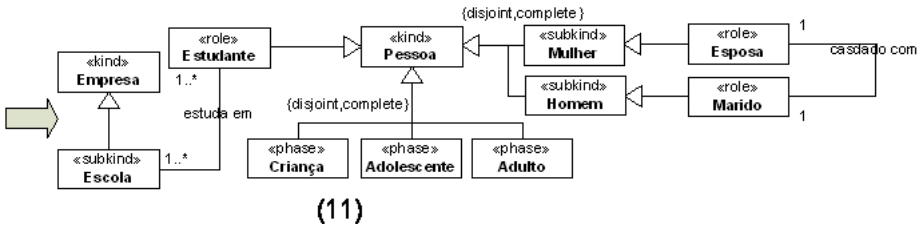

Figura 21 – Passos de formação de um modelo OntoUML pela aplicação das Regras de Construção propostas.

3.3. MODELO DE RACIOCÍNIO SOBRE PADRÕES DE PROJETOS - DPMODEL

Foram apresentados, neste trabalho, os padrões de projetos e os papéis que as entidades podem atuar nestes. Entretanto o metamodelo da OntoUML não representa diretamente essas informações, sendo necessário pesquisas complexas em OCL ou SQL para extrair estas informações.

Para contornar este problema é proposto um modelo para representar essas informações acerca de um modelo implementado em OntoUML. Esse modelo proposto – doravante

denominado DPModel – não tem por objetivo substituir o metamodelo da OntoUML, e sim fornecer outra visão do modelo expresso em OntoUML, realçando os padrões de projetos existentes.

O DPModel de um modelo em OntoUML possibilitará utilizar as informações para inferências sobre os estereótipos de um tipo, sugerir conceitos para complementar um padrão de projeto, ou mesmo, após os adendos propostos no próximo capítulo, explicar a outro modelador o *rationale* do modelo.

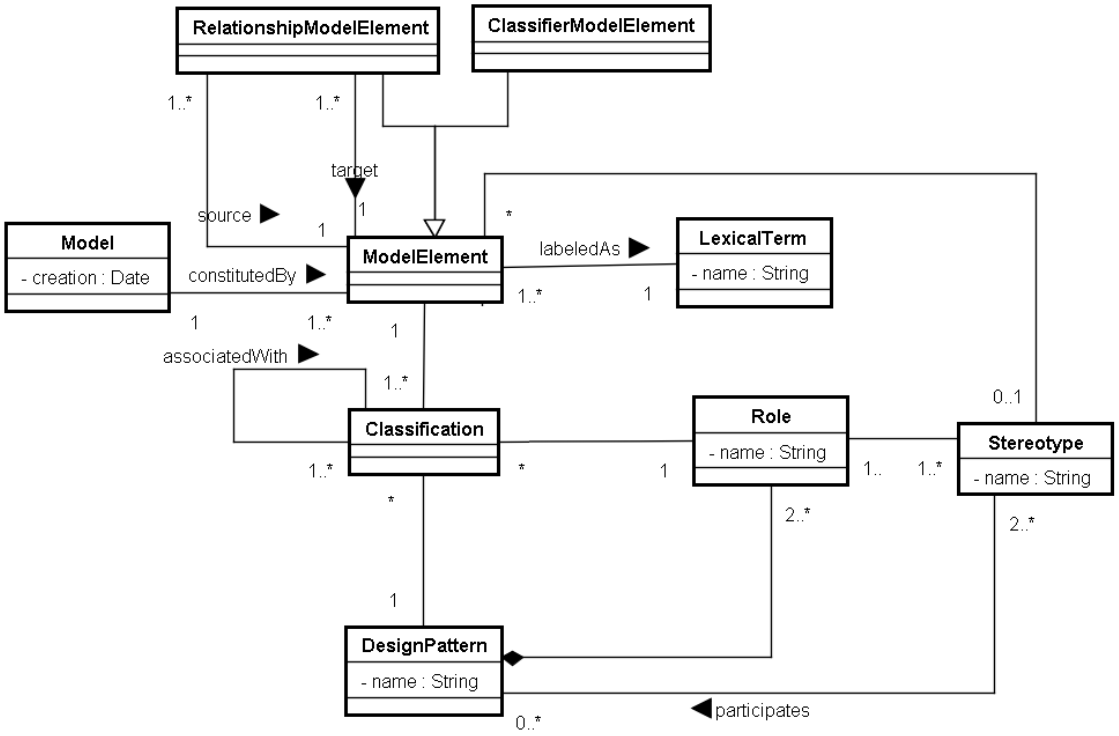

Figura 22 - Modelo para raciocínio sobre os Padrão de Projetos utilizados na OntoUML.

Os conceitos do metamodelo são, conforme é possível visualizar na Figura 22:

- *Model*: Representa um modelo específico. Um modelo pode ter um autor e uma data de criação. Um *Model* é constituído de *ModelElements*. É criada uma instância para cada modelo em OntoUML.
- *ModelElement*: um *ModelElement* é um elemento de um modelo. Pode ser a representação tanto de um relacionamento como a representação de um classificador. Um ModelElement pode ser estereotipado e também tem um

rótulo associado a ele. É criada uma instância para cada instância de Classifier ou Relationship da OntoUML.

- *LexicalTerm*: termo léxico que nomeia um tipo.
- *Classification*: classifica um elemento de modelo, como executando um papel, dentro de um padrão de projetos. Elementos que participam de um mesmo padrão de projeto são ligados pelo relacionamento *associatedWith*.
- *Role*: representa papéis possíveis de serem executados por um *ModelElement* no contexto de um Padrão de Projeto.
- *Stereotype*: representa os estereótipos possíveis. No caso do fragmento da OntoUML utilizado neste trabalho: *Role, Phase, SubKind* e *Kind*.
- *Design Pattern:* os padrões de projetos existentes.

Alguns conceitos do DPModel são preenchidos com informações estáticas para um dado metamodelo, no caso OntoUML. Para estes conceitos não são necessárias inserções ou alterações nas instâncias para cada modelo criado. Estes conceitos são: os *Design Patterns* que existem no metamodelo, no caso {*PhasePattern; SubKindPattern; RolePattern*}; os papéis que podem ser exercidos pelos elementos do modelo {*PhasedSortalDPR; PhaseDPR; SubKindSuperTypeDPR; SubkindDPR; RoleSuperTypeDPR; RoleDPR; RoleTargetDPR};* e os estereótipos existentes no metamodelo {*Kind, SubKind, Role, Phase*}.

Entretanto em sua maioria os elementos do DPModel são informações extraídas da representação do modelo em OntoUML. Segue abaixo as regras de derivação para os conceitos do modelo:

- *Model*: É criada uma instância para cada modelo OntoUML.
- *ModelElement*: É criada uma instância para cada instância de *Classifier* ou *Relationship* em um modelo OntoUML. É associado pela relação *constitutedBy* ao seu respectivo Modelo.
- *LexicalTerm*: Representa um termo léxico associado a um elemento de modelo. É criada uma instância para cada nome distinto que representa um tipo.
- *Classification*: Classifica um elemento de modelo, como executando um papel, dentro de um padrão de projetos. Regras:

a) *PhasedSortalDPR:* alvo de um RelationshipModelElement que esteja estereotipado como Generalization e cuja fonte seja um ModelElement estereotipado como *Phase*.
b) *PhaseDPR: ModelElement* estereotipado como *Phase*.
c) *RoleSuperTypeDPR:* o alvo de um *RelationshipModelElement* que esteja estereotipado como *Generalization* e cuja fonte seja um *ModelElement* estereotipado como *Role*.
d) *RoleDPR: ModelElement* estereotipado como *Role*.
e) *RoleAssociationDPR: RelationshipModelElement* que tenha sido capturado pela interação do guia metodológico, que expresse a dependência relacional de um tipo estereotipado com *Role.*
f) *RoleTargetDPR: ModelElement* associado por um *RelationshipModelElement* classificado como *RoleAssociationDPR*.
g) *SubKindSuperTypeDPR: ModelElement* alvo de um *RelationshipModelElement* que esteja estereotipado como Generalization e cuja fonte seja um ModelElement estereotipado como *SubKind*.
h) *SubkindDPR: ModelElement* estereotipado como *SubKind*.

3.3.1. UTILIZAÇÃO DO MODELO PARA SUGESTÕES A RESPOSTAS DO GUIA METODOLÓGICO

O DPModel apresenta informações importantes sobre os modelos e capacita uma ferramenta automatizada baseada nesse modelo a utilizar estas informações na criação de outros modelos. Essas informações podem ser utilizadas para sugerir respostas às perguntas demonstradas no guia metodológico, como as listadas a seguir:

1. **Superconceito mais usuais de um conceito**: Pesquisar em todos os modelos os elementos que tenham o mesmo nome e quais são os nomes mais usuais do seu supertipo.
2. **Conceitos complementares mais usuais de uma partição**: obter todos os *ModelElements* associados ao *LexicalTerm* pesquisado e depois obter todos os *ModelElements* que estejam associados com esses pela relação *associatedWith* tipo *PhasePattern* ou *SubKindPattern* dependendo do papel do conceito pesquisado.

3. **Relacionamento de dependência mais comum de um conceito estereotipado como « *Role* », dado um « *Role* » e o seu Supertipo**: obter todos os *ModelElements* associados ao *LexicalTerm* pesquisado, que estejam executando o papel « *Role* » e obter o *ModelElement* associado ao conceito pesquisado e que tenha o papel *RoleTarget*.

Por exemplo, ao obter sugestões por meio da Regra (3) do conceito *Marido,* subtipo de *Homem,* é possível oferecer ao modelador que na maioria dos casos é associado com *Mulher* através do relacionamento *CasadoCom*.

3.3.2. DEMONSTRAÇÃO DOS DADOS CAPTURADOS COM O DPMODEL

O objetivo desta seção é demonstrar as informações capturadas com o DPModel e como essas informações podem ser utilizados como sugestões para responder o guia metodológico. O DPModel provê uma visão diferente do modelo realçando os padrões de projetos utilizados. Entretanto, as informações representadas no DPModel não devem requerer mais interação com o modelador, e sim recuperar essa informação da representação padrão modelo e das interações do modelador com o guia metodológico. Para exemplificar as informações capturadas pelo DPModel, será utilizado o exemplo da seção 3.2.

Na criação do modelo, é instanciado o conceito *Model,* com a data 25 de Agosto de 2010. Para todos os elementos inseridos no modelo, são criados conceitos *ModelElement* que são relacionados ao *Model* pela propriedade *constitutedBy*. Segue a análise dos conceitos na ordem de inserção destes no exemplo da seção 3.2.

O primeiro conceito a ser analisado é o *Phase* Criança, subtipo do *Kind* Pessoa, como discutido um *Phase* só se manifesta como uma partição. A partição é composta pelos conceitos ⟨Criança, Adolescente, Adulto ⟩. Os conceitos que fazem parte da partição são classificados com o papel *PhaseDPR,* o supertipo da generalização, o conceito Pessoa, é classificado com o papel *PhasedSortalDPR*.

O conceito Escola é estereotipado como *SubKind,* e tem como supertipo o conceito Empresa com o estereótipo *Kind*. Portanto o conceito Escola é classificado com o Role *SubkindDPR* o conceito Escola é classificado com o conceito *SubKindSuperTypeDPR.*

O tipo Estudante estereotipado como *Role* logo é classificado como *RoleDPR*, pela regra (d). Estudante é subtipo de Pessoa. Logo Pessoa é classificado como *RoleSuperTypeDPR* pela regra (c). No exemplo Estudante é um papel desempenhado por uma Pessoa quando associada a uma escola pelo relacionamento "estudaem", este relacionamento foi capturado pela interação do guia metodológico com o usuário, portanto pela regra (e) este relacionamento é classificado como *RoleAssociationDPR* e pela regra (f) o conceito Escola é classificado como *RoleTargetDPR* pela regra (g).

O conceito Mulher é estereotipado como Subkind, portanto é classificado como um *SubkindDPR* pela regra (h). Mulher é subtipo de Pessoa, portanto este é classificado como *SubKindSuperTypeDPR* pela regra (g). Mulher tem como conceito complementar o conceito Homem, este também estereotipado como *SubKind*, logo também classificado pela regra (h) como *SubkindDPR*.

Por fim, é inserido o tipo Marido estereotipado como *Role* e pela regra (d) classificado como *RoleDPR*. Marido é subtipo de Homem, portanto homem é classificado pela regra (c) como *RoleSuperTypeDPR*. O guia metodológico também captura que um Homem se torna um Marido quando associado pelo relacionamento "casadocom" com o tipo Esposa. Logo o relacionamento é classificado como *RoleAssociationDPR* pela regra (f) e o tipo esposa como *RoleAssociationTarget (g).* O tipo Esposa também tem o estereotipo Role, portanto classificado pela regra (d) como *RoleDPR*. Esposa é subtipo de Mulher, que pela regra (c) é classificado como *RoleSuperTypeDPR*. Pelo guia metodológico capturamos que uma mulher só é Esposa quando também associada pelo relacionamento "casadocom" com um Marido. Logo Marido é classificado como *RoleTargetDPR*.

Segue abaixo uma tabela com as classificações de todos os tipos utilizados no exemplo.

Tabela 5 - Classificação dos conceitos

Tipo	**Padrões de Projeto no qual o tipos se manifestam.**	**Papéis executados no padrão de projeto.**
Criança	*PhasePattern*	*PhaseDPR*
Adolescente	*PhasePattern*	*PhaseDPR*
Adulto	*PhasePattern*	*PhaseDPR*

Pessoa	*PhasePattern*	*PhasedSortalDPR*
	RolePattern	*RoleSuperTypeDPR*
	SubKindPattern	*SubKindSuperTypeDPR*
Escola	*SubkindPattern*	*SubkindDPR*
	RolePattern	*RoleTargetDPR*
Empresa	*SubkindPattern*	*SubKindSuperTypeDPR*
Estudante	*RolePattern*	*RoleDPR*
estudaem	*RolePattern*	*RoleAssociationDPR*
Mulher	*SubKindPattern*	*SubkindDPR*
	RolePattern	*RoleSuperTypeDPR*
Homem	*SubKindPattern*	*SubkindDPR*
	RolePattern	*RoleSuperTypeDPR*
Marido	*RolePattern*	*RoleDPR*
	RolePattern	*RoleTargetDPR*
Esposa	*RolePattern*	*RoleDPR*
	RolePattern	*RoleTargetDPR*
casadoCom	*RolePattern*	*RoleAssociationDPR*

3.4.CONCLUSÃO

O Capítulo apresentou um guia metodológico para construção de modelos. Esse guia baseia-se no fato da OntoUML ser uma linguagem de padrões de projetos. Como demonstrado, a utilização de padrões de projeto facilita a atividade de modelagem, pois, como todas as técnicas de reuso, impedindo que erros comuns sejam cometidos evitando desta maneira o retrabalho na construção de uma ontologia de domínio. Além disso, facilita o entendimento do modelo, pois o *rationale* de um padrão de projetos usualmente é mais explicito ou é de conhecimento prévio de outros modeladores.

Como parte do guia metodológico, este capítulo propõe utilizar os padrões de projetos que se manifestam na OntoUML para construir regras de formação de modelo. Essas regras de formação sistematizam a construção do modelo, dessa modo a possibilitar a criação de uma ferramenta automatizada para guiar o modelador no processo de criação de modelos conceituais.

Também foi proposto um modelo conceitual para representar os padrões de projetos utilizados em um modelo, denominado DPModel. O DPModel não objetiva substituir a representação do modelo em OntoUML e sim dar outra visão ao modelo, com ênfase nos padrões de projetos manifestos. As informações presentes no DPModel são utilizadas para a criação de uma base histórica com o objetivo de criar sugestões para os usuários. Essas sugestões são utilizadas para responder as perguntas do guia metodológico.

CAPÍTULO 4. SUPORTE AO MODELADOR PARA ESTEREOTIPAÇÃO DE CONCEITOS EM ONTOUML

Este capítulo demonstra um guia metodológico para apoiar o modelador na escolha dos estereótipos a serem utilizados para representar um tipo. Cada estereótipo representa metapropriedades diferentes, busca-se, neste capítulo apresentar maneiras indiretas de capturar estas metapropriedades e auxiliar o modelador na escolha do estereótipo a ser utilizado.

A OntoUML é uma linguagem fruto da revisão da linguagem UML feita pela UFO. Consequentemente é uma linguagem ontologicamente bem fundamentada. Essa revisão atacou principalmente problemas de incompletude ontológica. Deste modo, foram incluídos mais construtores para aumentar a expressividade da linguagem.

Entretanto, a adição desses construtores à linguagem aumentou a complexidade desta, pois existem vários estereótipos possíveis para um tipo e a escolha de qual estereótipo utilizar e esta atividade não é trivial para um modelador inexperiente. Além disso, um modelador inexperiente que não está completamente familiarizado com os estereótipos da linguagem pode não compreender totalmente as implicações das distinções ontológicas existentes.

No capítulo anterior, foi apresentado um guia metodológico automatizado, que uma vez escolhido o estereótipo de uma classe, ajuda o modelador a construir o modelo através da instanciação de padrões resultantes da estereotipação. Este capítulo apresenta um adicional a este guia metodológico, ou seja, auxilia o usuário inexperiente na escolha desses estereótipos que iniciam o processo de instanciação de um padrão de projeto.

Adicionalmente, é demonstrado que, por meio da interação com o modelador consegue-se capturar o *rationale* para a escolha do estereótipo. Desse modo, um modelador inexperiente pode se beneficiar com o *Design Rationale* no processo de construção de modelos, pois é possível que o modelador recupere o motivo que levou um outro modelador a fazer as suas escolhas de modelagem.

4.1. SUPORTE À ESCOLHA DOS ESTEREÓTIOPOS

Como apresentado, os estereótipos da OntoUML refletem as distinções ontológicas presentes na UFO. Logo, os vários classificadores apresentam valores diferentes para as metapropriedades de um tipo. No escopo deste trabalho, as principais metapropriedades presentes nos construtores são:

- Princípio de Identidade
- Princípio de Rigidez
- Dependência Relacional

Assim, o problema de escolha de qual estereótipo utilizar para representar um conceito é reduzido ao problema de escolher o valor dessas metapropriedades. Entretanto entender a definição dessas metapropriedades e aplica-la a um conceito pode ser um trabalho árduo para um projetista não acostumado a linguagem.

A fim de reduzir esse esforço, é proposta uma maneira indireta de inferir o valor das metapropriedades de um conceito, por meio de perguntas de fácil compreensão e respostas que não demandem do modelador o entendimento das distinções ontológicas contidas na linguagem. Em outras palavras, a ideia é verbalizar as consequências das metapropriedades quando aplicadas a um conceito.

Para citar um exemplo, ao invés de perguntar diretamente ao modelador se o conceito Pessoa é um tipo rígido, a pergunta feita pela ferramenta seria: Uma Pessoa pode deixar de ser uma Pessoa? Uma Pessoa pode não ter sido um Pessoa no passado?

É importante ressaltar que essas classificações não são estáticas para uma palavra ou para um conceito. As classificações dependem da visão de mundo do projetista, do domínio sendo modelado e da definição do conceito ao qual a palavra representa. Por exemplo, analise estas duas frases:

1. A pessoa que comprou esta casa mora em Londres;
2. Ele não é mais uma **pessoa,** é um **monstro**.

No primeiro exemplo o conceito **pessoa** visto como criatura humana, é uma classe rígida. A pessoa que comprou sua casa sempre será uma pessoa.

$$\{w, w'\} \in W \, , \{\neg \exists x | \, \mathrm{p(x, w)} \wedge \sim \mathrm{p(x, w')}\}$$

Porém, no segundo caso, o conceito pessoa está fazendo uma transição para fora do conjunto pessoa.

$$\{w, w'\} \in W, \{\exists x | \, \mathrm{p(x, w)} \wedge \sim \mathrm{p(x, w')}\}$$

O exemplo acima é um caso de polissemia, quando uma palavra contém dois significados distintos. No primeiro caso pessoa significa criatura humana, no segundo se dá como antítese ao conceito monstro que tem o significado de pessoa cujas características humanas (morais ou estéticas) estejam desfiguradas. Logo para classificar corretamente um conceito é necessário compreender qual a definição atribuída ao conceito que se pretende representar.

Na Figura 23 é apresentado um fluxograma das decisões das metapropriedades, apresentando a ordem em que serão feitas as perguntas para decidir o construtor do conceito.

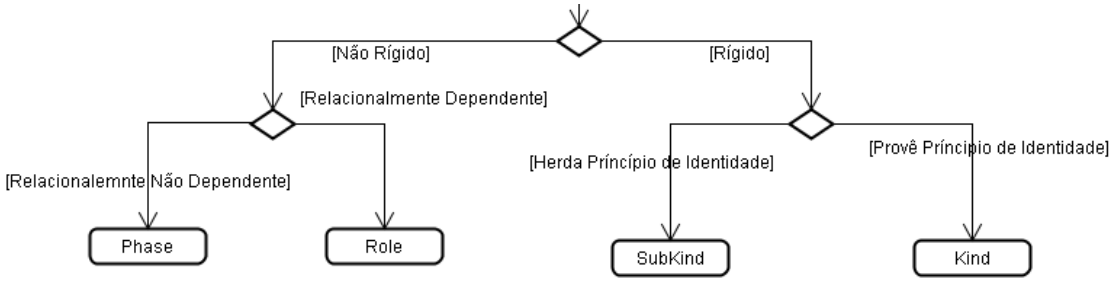

Figura 23 – Fluxograma de Decisão da metaclasse.

A respeito da rigidez do objeto, são possíveis apenas dois estados no fragmento da OntoUML apresentado: Rígido ou Antirrígido. Evidências de que um conceito não é rígido são exemplos de raciocínios que mostram um objeto **y** se tornar instância ou deixar de ser instância de um tipo X. Exemplos: Ele se tornou um **marido**; Ele era um **professor**. Ele se transformou em um **monstro**. Ele não é mais um **adolescente**. Todos esses exemplos demonstram que as classes marido, professor, monstro e adolescente não são rígidos (naquelas conceituações em que essas frases são admitidas). Logo estruturas linguísticas na forma "não é mais **X**" ou "se tornou **X**", onde **X** é o conceito ao qual inquirimos o estereótipo, evidencia uma característica não rígida do conceito. Perguntas:

- É aceitável um **X** deixar de ser um (a) **X**?
- É possível que uma instância não sendo um(a) **X**, se torne um(a) **X** ?

Caso já se tenha informações acerca da hierarquia do tipo é possível utilizar esta informação na pergunta. Por exemplo, caso tenha-se o tipo Marido como Subtipo de Pessoa.

- É possível uma Pessoa tornar-se um Marido? Ou é possível uma Pessoa que é um Marido deixar de ser um Marido?

A decisão quanto à característica de um conceito ser relacionalmente dependente, se dá por evidências de que um objeto *x* só pode instanciar um tipo **X** utilizando-se do relacionamento com outra entidade *y*.

Exemplos: João é um estudante por que está matriculado na Universidade. Jamile é esposa por que está casada com o Homem Pedro. Tais exemplos demonstram a dependência de classe, pois existe a necessidade de uma instância de Universidade ao qual João esteja matriculado para que ele seja considerado um estudante, do mesmo modo que é necessária a existência do homem ao qual Jamile seja casada para que ela seja considerada uma esposa.

Pergunta:

- Para que seja considerado um Estudante, é necessário um relacionamento com outra entidade?

O último principio da árvore de decisão distingue quanto a um conceito prover ou não a característica de identidade. É necessário notar que não estamos falando se o conceito tem ou não um princípio de identidade, mas se o princípio de identidade é provido pelo conceito analisado.

Um exemplo de princípio de identidade para uma pessoa seria a impressão digital. As classes Homem e Mulher também contem este princípio de identidade, porém esta característica de identidade é claramente pertencente à classe Pessoa. Logo, o conceito Pessoa provê um princípio de identidade, já as classes Homem e Mulher herdam esse princípio de Pessoa.

Pergunta:

- O princípio de identidade (exemplo de princípios de identidade são impressão digital ou carteira de identidade para pessoas, um número serial para máquinas, etc.) utilizado por esse tipo é herdado de algum supertipo?

As decisões tomadas pelo usuário serão armazenadas no modelo demonstrado na seção 4.2, e essas decisões além de serem utilizadas para inferir o estereótipo dos modelos, explicitam o *rationale* da escolha do estereótipo.

4.2. UTILIZAÇÃO DE ITENS LÉXICOS PARA METAPROPRIEDADES

O guia metodológico com perguntas torna muito mais intuitiva a escolha das metapropriedades. Entretanto, é possível que o modelador fique em dúvidas acerca do domínio. Ou seja, são admitidos aqui casos em que o modelador não é capaz de discernir se é possível alguém se tornar ou deixar de ser uma instância de uma máquina. Para esses casos entra em cena o subsistema de padrões léxicos.

Esse subsistema utiliza o nome do conceito para buscar padrões textuais que evidenciem situações nas quais este tenha um valor específico de uma metapropriedade. Dessa maneira, o projetista pode ter uma visão do conceito no domínio que ele está modelando com relação a essas metapropriedades.

Neste trabalho, a proposta é capturar os padrões textuais das metapropriedades rigidez e dependência relacional. Para utilizar padrões textuais, é necessário um corpo de texto suficientemente grande para ser possível encontrar resultados acerca de todos os domínios modelados. Assim, para conseguir dados suficiente,s foi utilizado como corpo de texto o maior repositório de conhecimento humano atualmente, a Web. Para utilizar a Web como corpo de texto foi feita a integração com a ferramenta de busca Google.

A decisão de utilizar a Web como corpo de texto apresenta algumas desvantagens, pois nem sempre a visão apresentada nos textos corresponderá à visão de mundo adotada pelo projetista. Este fato é muito acentuado visão imprecisa e emocional, contido em parte dos textos da internet, que não condizem com a abordagem usualmente requerida na construção de uma ontologia, por exemplo, é comum em textos se uma pessoa trabalha muito dizer “ele se tornou uma máquina”, ou se o seu o marido de uma mulher se tornou violento, ela dizer “ele não é mais uma pessoa é um monstro”.

Porém, o maior obstáculo deste método é encontrar padrões textuais que justifiquem com uma alta taxa de probabilidade, uma relação ontológica. A linguagem natural não respeita a maioria dos princípios buscados nas linguagens, pois, frequentemente, é ambígua e não lacônica, ou seja, é possível representar o mesmo conceito de várias

maneiras possíveis. Por outro lado, frequentemente é também não lúcida, o que é denotado pelo uso de figuras de linguagem, por exemplo: metáfora, alegoria e ironia. Ou seja, uma palavra pode ter vários significados. (GUIZZARDI, 2005)

De todo modo, é importante ressaltar que a decisão final de aceitar uma recomendação de estereotipação feita pelo sistema é sempre do usuário. Além disso, este mecanismo é utilizado pelo sistema sempre em casos excepcionais, ou seja: (i) quando o estereótipo não for implicado pela instanciação de um padrão em que o conceito pertence; (ii) quando o usuário for incapaz de responder às questões da seção anterior acerca das metapropriedades consideradas.

4.2.1. PADRÕES TEXTUAIS

A utilização de padrões textuais para inferência de metapropriedades já foi abordado por vários pesquisadores entre eles (VÖLKER, VRANDECIC, *et al.*, 2008)(CIMIANO, HANDSCHUH e STAAB, 2004). A principal ideia de qualquer abordagem por padrões textuais é a possibilidade de justificar uma relação ontológica com alguma precisão quando reconhecemos certas expressões idiomáticas / sintáticas / semânticas (CIMIANO, HANDSCHUH e STAAB, 2004).

É possível detectar essas expressões, por exemplo, no conjunto de perguntas que verbalizam as metapropriedades utilizadas na definição do conceito na seção 4.1. Uma das perguntas acerca da metapropriedade rigidez é: "É aceitável um **X** deixar de ser um (a) **X**?". Podemos evidenciar a expressão "deixar de ser um X", onde X é um tipo, em várias construções verbais que demonstram a não rigidez do tipo. Por exemplos: "deixar de ser um Marido", "deixar de ser uma Pessoa". As expressões encontradas no conjunto de perguntas propositalmente utilizam-se dessas expressões que evidenciam um fenômeno que justifica o valor da metapropriedade.

Entretanto esses padrões nem sempre justificam o valor de uma metapropriedade. Foram escolhidos padrões textuais que quando encontrados em uma frase frequentemente justificassem uma metapropriedade ontológica. Esta precisão foi aferida de maneira empírica, ou seja, os padrões textuais demonstrados neste trabalho foram os que apresentaram melhores resultados após vários testes. Este trabalho de padrões

textuais foi efetuado somente no idioma inglês devido a maior disponibilidade de Part-of-speech Tagger (POS Tagger) naquele idioma.

O POS Tagger é uma ferramenta capaz de criar marcações, definindo a função das palavras. Neste caso a função gramatical. Esta ferramenta filtra alguns resultados que são detectados erroneamente. Por exemplo, quando procuramos pelo termo *Computer* poderíamos encontrar o termo *Computer Hacker*. Este assunto será abordada na seção 5.1

Para demonstrar a não rigidez de um tipo – definição das metapropriedades na seção 2.1.5 – é necessário demonstrar que um indivíduo deixou de ser instância de um tipo. Por exemplo:

> ***Você não é mais um empregado**, você pode ser meu parceiro! Quando você se inscrever, você está se tornando sócio de uma empresa...*

Este texto demonstra a não rigidez do conceito empregado, pois um indivíduo representado pelo termo "Você" deixou de ser uma instância de empregado. Os padrões textuais escolhidos para demonstrar não rigidez foram:

- *is no longer a* **type.name**
- *became a* **type.name**
- *while being a* **type.name**

Esses padrões textuais justificam a não rigidez de um tipo. Entretanto, cada frase escolhida deve passar pelo crivo do projetista, pois é necessário que este julgue se a frase escolhida reflete o domínio modelado, pois, como discutido na seção 4.2, na linguagem natural, um termo pode se referir a vários conceitos diferentes.

Não foi possível encontrar um padrão textual que representasse a dependência relacional com a mesma precisão da metapropriedade rigidez. Contudo, o padrão que apresentou melhores respostas foram os apresentados abaixo. Com o crivo do modelador, para determinar se a frase escolhida é relacionada à dependência relacional, eles podem apresentar boas justificativas para a escolha da metapropriedade.

- *cannot be a* **type.name** *without*
- *cannot be a* **type.name** *unless*

Os padrões textuais escolhidos demonstram apenas duas das três metapropriedades utilizadas neste fragmento da OntoUML, além disso, só demonstra um valor para as metapropriedades. Devido à dificuldade de encontrar padrões que justifiquem o valor de uma metapropriedade não foi possível abordar todos os casos.

4.3. CAPTURA DE DESIGN RATIONALE

O conjunto de perguntas que verbalizam as metapropriedades utilizadas na definição do conceito (seção 4.1), além de auxiliar o modelador no processo de criação da ontologia nos fornece informações importantes que explicam as decisões do modelador quanto aos estereótipos atribuídos às entidades do modelo.

Neste trabalho, essas informações deverão ser utilizadas para gerar uma instância DPModel e, consequentemente, capturar o *design rationale* de um modelo, facilitando, desta maneira, o reuso e a compreensão do modelo por outros modeladores. Para representar essas informações adicionais, o modelo DPModel foi estendido, como pode ser visualizado na Figura 24 (classes adicionadas na área em realce).

As classes adicionadas ao modelo foram: *Stereotyping; Justification*; *Lexical Pattern Verbalization*; *Metaproperty; Person*; Como é possível visualizar no modelo, é capturado o autor da justificativa e dos criadores das instâncias, criando uma rastreabilidade das operações no modelo.

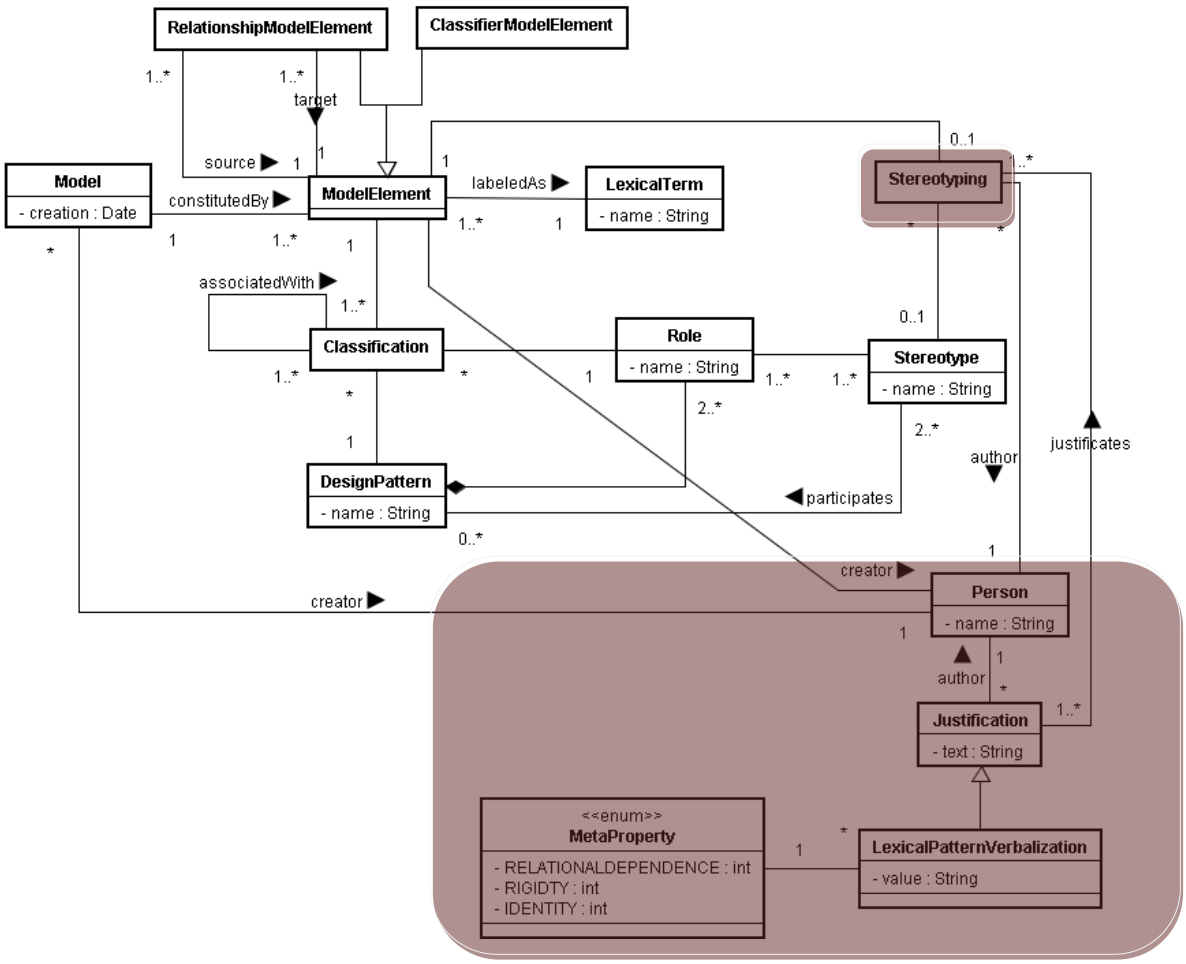

Figura 24 – Modelo completo com classes para Justificação de Ações

A Classe *Lexical Pattern Verbalization* representa as justificativas capturadas para o valor das metapropriedade. Como apresentado, foram propostas perguntas indiretas que capturam o valor das metapropriedades, logo é possível persistir a pergunta e a resposta como justificativa para a escolha.

Por exemplo, considere o modelo na Figura 25, no qual se busca auxiliar a estereotipação do tipo Pai. A interação seria:

Q) É possível uma pessoa que não é Pai se tornar um Pai?

R) Sim

Logo é possível criar uma instância do tipo *Lexical Pattern Verbalization,* relacionada com a Metapropriedade Rigidez e o valor do atributo *value* é verdadeiro. O autor seria quem respondeu a pergunta e o valor do atributo texto seria: “É possível uma pessoa que não é pai se tornar um Pai”.

Outra fonte de justificativas é o apoio por padrões textuais, quando o modelador sinalizar que uma frase corresponde à visão de mundo do seu domínio, esta frase também será adicionada como justificativa à sua decisão pelo valor da metapropriedade.

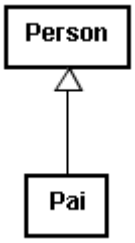

Figura 25 - Exemplo para captura de Rationale

4.3.1. UTILIZAÇÃO DO MODELO PARA SUGESTÕES

O Modelo para raciocínio sobre padrões de projetos, DPModel, apresentado na Figura 24, representa qual é o estereótipo de um elemento de modelo e quais os papéis que o conceito está desempenhando naquele modelo.

Utilizando esse modelo proposto é possível extrair informações importantes acerca das ontologias para serem utilizadas durante o processo de modelagem. Esse histórico pode ser empregado para mostrar sugestões sobre: a) estereótipos de tipos; b) valor de metapropriedades de um conceito; c) apresentar a justificativa das decisões passadas. Exemplos de questões que podem ser respondidas por consultas ao DPModel incluem:

1. **O Estereótipo mais usual, para um conceito que representa um termo léxico**: obtido pesquisando todas as instâncias de *ModelElement* associadas a este termo léxico e obtendo o estereótipo associado a este *ModelElement*. Assim se existem dez modelos que o item "Person" aparece e em oito modelos ele está estereotipado como *Kind* e em dois como *SubKind*, podemos falar que em 80% dos casos o conceito "Person" é estereotipado como *Kind.*

2. **Estereótipo mais usual para um conceito representando um papel específico** : Como definido no capítulo 3, elementos podem exercer diferentes papéis em um padrão de projeto. Por exemplo, na Figura 26, um *MusicalInstrument* é o alvo da depêndecia relacional do conceito *Musician.* Uma Pessoa(*Person)* só pode ser um Músico(*Musician)* se tocar ao menos um Instrumento Musical (MusicalInstrument), consequentemente exerce o papel de *RoleTargetDPR*.

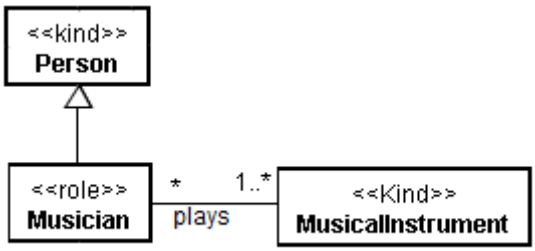

Figura 26 – Exemplo de aplicação de papéis no contexto de um Padrão de Projeto.

Alguns papéis exercidos são detectados sem a necessidade de conhecer o estereótipo do conceito. No exemplo da Figura 26, o estereótipo *Kind* não influencia no papel exercido pelo tipo no contexto do padrão de projeto *RolePattern*. Este é o exemplo do *PhasedSortalDPR, RoleSuperTypeDPR, SubKindSuperTypeDPR,* e *RoleTargetDPR*. Nesses casos, é possível obter uma sugestão histórica mais contextualizada para o modelador.

Para responder essa pergunta, é necessário obter todos os *ModelElements* associados ao mesmo *LexicalTerm* e que tenha um *Classification* associado ao mesmo *Role.*

3. **Valor de Metapropriedade para um conceito**: Como demonstrado, quando existe interação para auxiliar o modelador na escolha do estereótipo, são capturadas justificativas para cada valor de metapropriedade. Além disso, cada escolha de estereótipo também é uma escolha para o valor de metapropriedades. Logo, é possível fazer uma pesquisa com maior granularidade, e obter qual o valor mais usual para determinada metapropriedade.

Por exemplo, o conceito Pessoa, pode, em muitos modelos, ser representado como *Kind* ou *SubKind,* dependendo da posição na árvore taxonômica. Entretanto em quases todos os modelos o conceito será rigido.

4.4.CONCLUSÃO

Este capítulo propõe maneiras de auxiliar o modelador na escolha do estereótipo a um conceito. Essa escolha não é uma tarefa fácil para um modelador principiante, ou que não esteja completamente familiarizado com a linguagem OntoUML. Portanto, o capítulo se propõe a fornecer dois métodos distintos para oferecer apoio ao modelador.

O primeiro método busca, através de perguntas acerca do conceito, extrair o valor das metapropriedades desse conceito. Tais metapropriedades tornam possível fazer as distinções ontológicas propostas na linguagem e escolher corretamente o estereótipo a ser utilizado.

Em alguns casos pode ocorrer do modelador não conseguir responder às questões acerca de um conceito, ou simplesmente ter dúvidas acerca do melhor estereótipo. Para esses casos, pode-se recorrer a uma base histórica. Essa base histórica tem acesso a decisões de estereotipação dos modeladores e até da justificativa para essas escolhas.

Por fim, além de propiciar acesso ao conhecimento coletivamente construído de uma comunidade de modeladores em uma base histórica, a abordagem proposta aqui dá suporte a uma outra fonte de conhecimento a fim de auxiliar o modelador inexperiente em tarefas de estereotipação: a web. Quando todos os outros mecanismos de suporte falham, a abordagem aqui proposta dá suporte à submissão dos padrões léxicos correlacionados a metapropriedades para pesquisa em uma máquina de busca. Os resultados retornados dessas pesquisas são, então, processados para que indícios de valores de metapropriedades possam ser derivados.

Os métodos aqui descritos visam facilitar o processo de modelagem tanto ao modelador inexperiente, quanto o experiente em momentos de dúvidas, no que tange a representação de alguns conceitos.

CAPÍTULO 5. CONSTRUÇÃO DE UMA FERRAMENTA DE APOIO A CRIAÇÃO DE ONTOLOGIAS DE DOMÍNIO

Este capítulo apresenta uma ferramenta para criação de ontologias de domínio em OntoUML: OntoUML WebEditor. Esse editor é uma ferramenta Web, capaz de criar modelos visuais em OntoUML. Serão abordadas neste capítulo, as tecnologias utilizadas para o desenvolvimento, arquitetura e as funcionalidades da ferramenta.

A criação de ontologia é uma tarefa complexa por definição, já que esta deve exprimir um consenso entre vários modeladores sobre um determinado domínio. Como discutido anteriormente, a utilização de uma linguagem ontologicamente bem fundamentada também se faz necessária para assegurar a qualidade das ontologias, entretanto, isso também pode aumentar a complexidade da tarefa para projetistas iniciantes no uso da linguagem.

A OntoUML contém inúmeras restrições e a checagem dessas restrições manualmente seria inviável. Como a maioria das linguagens, a criação de uma ferramenta de modelagem para OntoUML é o caminho usual para a sua adoção. Em (BENEVIDES, 2007), foi proposta uma ferramenta capaz de checar todas as restrições de OntoUML, evitando a criação de modelos sintaticamente incorretos em OntoUML.

Apesar da ferramenta proposta por (BENEVIDES, 2007) ser útil para checar as restrições, não foi projetada para auxiliar a tarefa de modelagem. A tarefa de estereotipar cada conceito e relação, ainda pertence ao modelador sem nenhum auxílio da ferramenta.

Dessa forma, apresenta-se, neste capítulo, uma ferramenta que, além de restringir modelos incorretos, apoia o modelador na tarefa de modelagem. Primeiramente, demonstra-se como a ferramenta auxilia o modelador durante o processo de estereotipação. Este auxílio é prestado pela implementação: (i) conjunto de perguntas; (ii) apresentação de frases buscadas por padrões textuais que demonstram indícios do valor da metapropriedade; (iii) sugestões das metapropriedades e estereótipos através da base histórica.

Como segundo método de apoio é demonstrada a implementação do guia metodológico proposto no capítulo 3. Esse guia utiliza os padrões de projetos detectados na linguagem para auxiliar a construção de modelos, também são apresentadas sugestões ao usuário baseadas na base histórica do DPModel.

Acredita-se também que a ferramenta proposta neste trabalho – OntoUML *WebEditor* – facilitaria a adoção da OntoUML por outros modeladores pois suaviza a curva de aprendizagem da linguagem. Outro ponto que contribui para a adoção desse editor é fato deste ser uma aplicação Web, podendo, assim ser disponibilizada para qualquer pessoa, sem necessidade de instalação de softwares localmente no computador. É possível visualizar a ferramenta executando em um navegador na Figura 27.

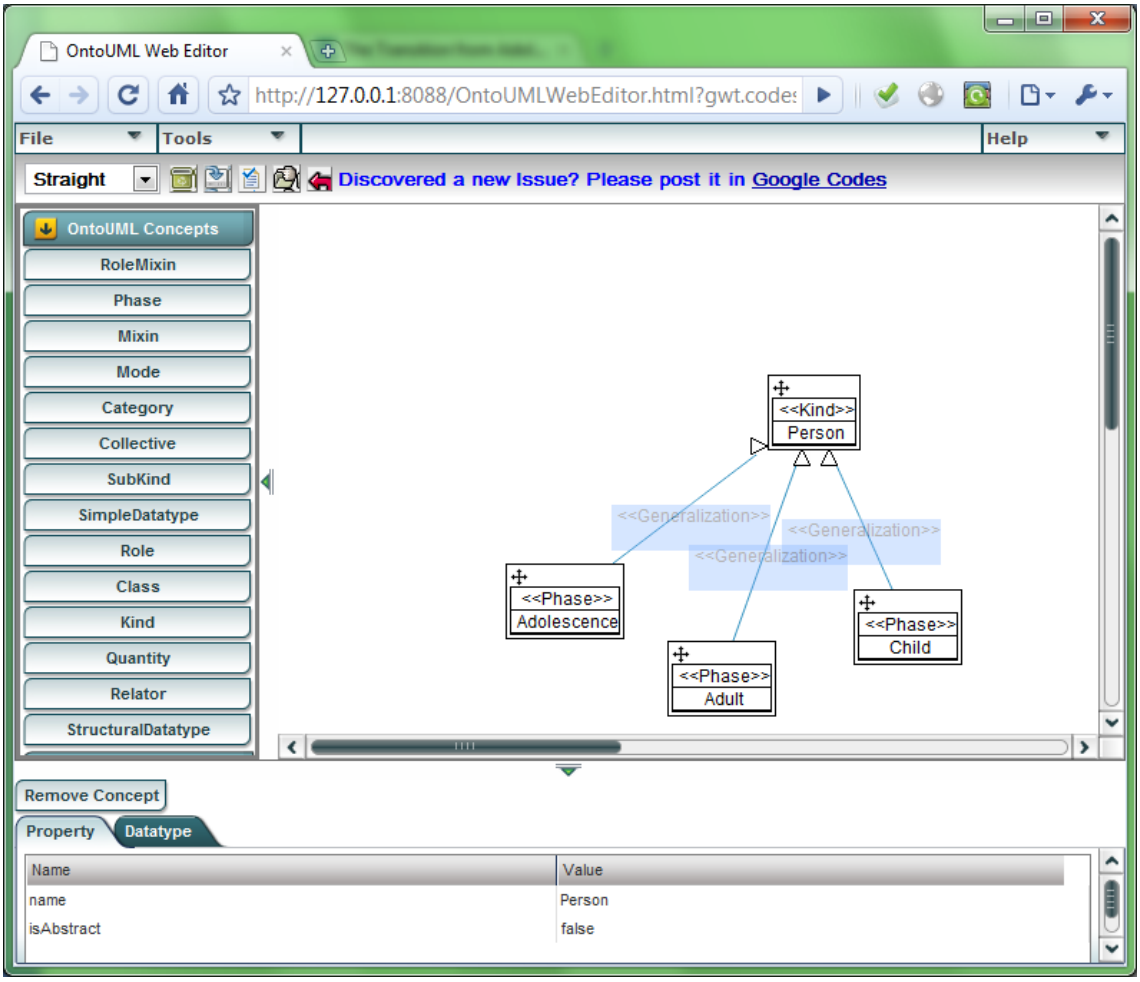

Figura 27 - Ferramenta de Modelagem OntoUML WebEditor

Este capítulo está organizado da seguinte maneira: na seção 5.1, está demonstrada a implementação de um assistente que utiliza o Guia metodológico proposto no capítulo 3 em conjunto com a proposta do capítulo 4 para auxiliar o modelador; Na seção 5.2, demonstra-se a implementação do DPModel, que é utilizado para: (i) apresentar sugestões ao modelador; (ii) capturar o *Design* Rationale da interação do modelador com a ferramenta. Na seção 5.3, é apresentada a implementação do sistema de padrões textuais, responsável por obter frases que manifestem um certo valor a uma metapropriedade e que auxiliem o modelador em sua escolha; Na seção **Erro! Fonte de eferência não encontrada.**, é demonstrada a Arquitetura da Ferramenta e as tecnologias utilizadas na implementação; Por último, na seção 5.5 são apresentadas as conclusões deste capítulo

5.1. GUIA METODOLÓGICO

O guia metodológico é responsável por auxiliar modeladores inexperientes no processo de criação de um modelo conceitual.Como discutido no capítulo 3, o objetivo é tornar a construção de modelos ontologicamente bem fundamentados um processo mais simples para desenvolvedores iniciantes.

A ferramenta apresenta os passos deste guia de maneira amigável ao usuário, apresentando, sempre que possível, sugestões históricas baseadas no DPModel. Além disso, a ferramenta busca utilizar o máximo de conhecimento disponível, evitando fazer perguntas que podem ser inferidas no modelo. O objetivo é auxiliar o usuário, com o mínimo possível de interações adicionais. Primeiramente são abordados, nesta seção, os métodos para prestar suporte ao modelador no processo de estereotipação dos conceitos propostos no capítulo 4.

Um modelador pode criar uma nova classe em sua ontologia clicando o botão “Class”, na tela exibida na Figura 27. Assim que isso é feito, o assistente é disparado para auxiliar o usuário na identificação do estereótipo e de um possível padrão de projeto em que se adeque.

A primeira pergunta deste assistente é “Qual o Nome deste Tipo?”. Em seguida, é apresentado um segundo diálogo com a pergunta é “[Tipo.nome][5] é subtipo de algum outro tipo?”. Estes passos nos fornecem informações básicas do tipo e localiza este tipo em uma taxonomia caso seja possível, o que fornece informações úteis que podem ser inferidas pelo relacionamento de generalização.

Após essas duas perguntas, segue-se o fluxo proposto na Figura 23 do CAPÍTULO 4. Logo, a primeira pergunta a ser respondida é acerca da rigidez do conceito. Como foi discutido, um dos objetivos é evitar interações desnecessárias com o usuário. Logo, antes de apresentar um diálogo para o usuário a ferramenta tenta inferir um valor para esta metapropriedade. Esta propriedade pode ser inferida utilizando algumas definições da OntoUML:

1. Sendo X subtipo de Y, se X é rígido logo Y é rígido. Logo se a classe analizada é supertipo de um tipo rígido ela deve ser necessariamente rígida. No fragmento da OntoUML deste trabalho esta pesquisa pode ser definida como:

   ```
   OCL:
   Classifier.allInstances()->select( x | x.allSuperTypes()->includes(self))->asSet()->exists(
   x | x.oclIsKindOf(RigidSortalClass))
   ```

2. Por outro lado, sendo X subtipo de Y se Y não é rígido logo X não é rígido. Portanto se algum dos supertipos da classe analizada não é rígido, este tipo deve ser obrigatoriamente não rígido. Esta regra pode ser definida como:

   ```
   OCL:
   self.allSuperTypes()->exists(x | x.oclIsKindOf(AntiRigidSortalClass))
   ```

Também são feitas pesquisas no DPModel em busca dos valores das metapropriedades do supertipo desta entidade.

Caso as pesquisas apresentadas acima não consigam inferir um valor para a metapropriedade rigidez é apresentado um diálogo com a pergunta acerca da rigidez do conceito.

---

[5] Para facilitar a explanação das perguntas serão utilizadas a notação: [Tipo.nome] Nome do tipo analizado; [Tipo.supertipo.nome] Nome do supertipo do tipo analizado.

Contudo para que esta pergunta se apresente ao usuário de maneira mais amigável, é verificado primeiramente se já existe algum supertipo a este tipo, caso exista a pergunta apresentada ao usuário será:

É possível que um [supertipo.nome] se torne um [tipo.nome]?Ou que uma [tipo.nome] deixe de ser um [tipo.nome]?

Por exemplo, para o tipo Marido e o seu supertipo Pessoa a pergunta será:

É possível que um Pessoa se torne um Marido?Ou Que um Marido deixe de ser Marido?

Caso não exista a informação hierárquica, a pergunta será:

É possível que um [tipo.nome] deixe de ser um [tipo.nome] ou que uma instância que não seja [tipo.nome] se torne um [tipo.nome]?

Por exemplo, para o conceito Esposa, sem nenhuma informação hierárquica a pergunta seria:

É possível que uma Esposa deixe de ser uma Esposa ou que uma instância que não seja do Tipo Esposa se torne uma Esposa?

Assim que a ferramenta obtém um valor para a metapropriedade rigidez ela utiliza este valor para escolher a próxima pergunta. Caso a resposta evidencie o comportamento rígido do tipo, será feita pergunta sobre o provimento do princípio de identidade. Caso contrário, a pergunta se referirá a dependência relacional.

Caso o conceito seja rígido, antes de perguntar acerca do provimento identidade, dentro do princípio de evitar interações desnecessárias, será checado se é possível inferir o valor desta metapropriedade. A OntoUML determina que um tipo pode herdar de apenas um tipo o princípio de identidade. Logo, se algum dos supertipos do tipo analisado já provê o princípio de identidade, o tipo analisado não pode prover outro princípio de identidade(GUIZZARDI, WAGNER, *et al.*, 2004).

A análise descrita é feita avaliando se já existe algum supertipo estereotipado como *Kind* ou se o usuário, através da interação com a ferramenta, informou explicitamente que algum dos supertipos provê o princípio de identidade.

```
OCL:
self.allSuperTypes()->exists(x | x.oclIsKindOf(Kind))
```

Caso não seja possível inferir o valor da metapropriedade, esta pergunta será feita:

> *Existe algum tipo de princípio de identidade provido para [tipo.nome]? Por exemplo. DNA, ou Documento de Identidade para Pessoa ou Números de Série para Objetos.*

No caso do tipo analisado não ser um tipo rígido, a ferramenta irá perguntar se o tipo tem algum relacionamento de dependência de classe com algum outro tipo. Caso haja conhecimento de algum supertipo, esta informação será utilizada na formulação da pergunta:

> *Deve haver alguma relação com outro tipo para que [supertipo.nome] seja considerado um [tipo]?*

No caso de tipo ser igual a Marido e supertipo Homem a pergunta seria formulada da seguinte maneira:

> P) *Deve haver alguma relação com outro tipo para que Homem seja considerado um Marido?*
>
> R) Sim.

Caso não haja nenhuma informação sobre a hierarquia do tipo a pergunta será:

> *Deve haver alguma relação com outro tipo para uma instância ser considerado um [tipo.nome]?*

Logo no caso de Marido a pergunta seria:

> P) *Deve haver alguma relação com outro tipo para uma instância ser considerada um Marido?*

Após a captura dessas informações, elas serão processadas e caso já exista informação o bastante para descobrir o estereótipo do tipo, este será atribuído a ele. Após descobrir o estereótipo de um conceito, a ferramenta continua a auxiliar o modelador utilizando o guia metodológico baseado em padrões apresentado no capítulo 3.

No caso do tipo analisado ter o estereótipo *SubKind,* conforme a Regra R_{SK}, caso a informação não esteja ainda representada no modelo, será perguntado ao modelador qual o supertipo. Afinal todo tipo deve herdar de um tipo *Kind*, direta ou indiretamente(GUIZZARDI, WAGNER, *et al.*, 2004). A próxima pergunta será se

existe algum tipo complementar a este tipo SubKind. No entanto, antes de efetuar a pergunta, a ferramenta procura primeiramente outros tipos complementares que já existam representados no metamodelo.

```
OCL:
   Generalization.allInstances()->select(z |
   Generalization.allInstances()->select(x |
   x.specific = self ).general->includes(z.general)
   and z.specific.oclIsKindOf(SubKind)).specific
```

Por exemplo, suponha um modelo no qual o tipo Parazoa, Subreino do Reino Animália é representado. Como demonstrado na Figura 28, uma vez representado o subKind Parazoa, o seguinte diálogo é estabelecido de forma automática com o modelador:

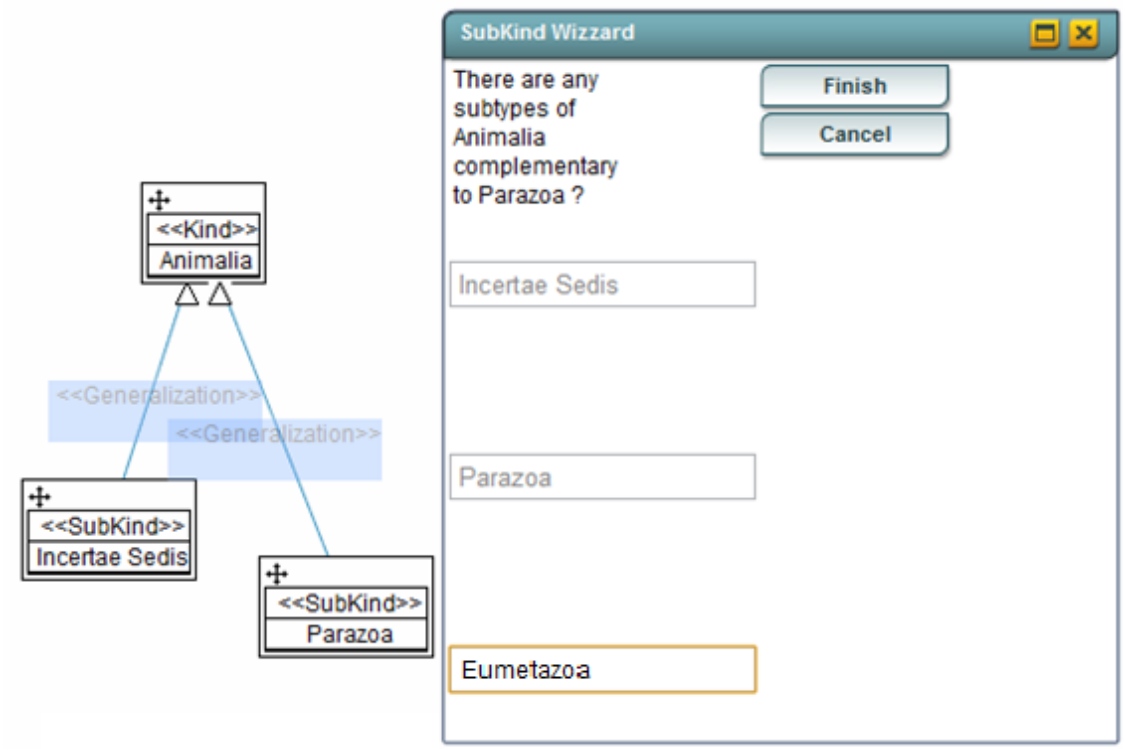

Figura 28 - Interface para capturar tipos complementares de um SubKind

Q) Existe algum subtipo de Animalia complementar a Parazoa?

R) {Incertae Sedi}, *Eumetazoa*

Caso o estereótipo do tipo analisado seja *Phase*, será requisitado ao modelador um supertipo para esta fase (caso este não esteja já representado no modelo), além de quais são as suas fases complementares (conforme a regra R_P, capítulo 3).

Caso o estereótipo analisado seja *Role,* a ferramenta apresentara ao modelador um diálogo (conforme a regra R_R, capítulo 3). Portanto seguem as perguntas:

1) Com qual tipo [supertipo.nome] deve estar relacionado para que seja considerado um [tipo.nome]?

2) Qual o nome do relacionamento que deve existir entre [tipo.nome] e [tipodependente.nome], para que [supertipo.nome] seja considerado um [tipo.nome]?

5.2. SUGESTÕES BASEADAS EM HISTÓRICO

O guia metodológico facilita o processo de criação de modelos conceituais por modeladores inexperientes. Entretanto como mais uma camada de apoio ao modelador a ferramenta busca utilizar outros modelos armazenados em seu repositório para apresentar sugestões às perguntas feitas pela ferramenta durante a execução do assistente.

O DPModel apresentado no capítulo 3 e estendido no capítulo 4, extrai dos modelos informações acerca dos padrões de projetos utilizados nos modelos e estas informações são utilizadas pela ferramenta para apresentar as sugestões.

Para utilização na ferramenta o DPModel, foi implementado em banco de dados relacional. Como discutido as informações deste modelo são em sua maioria derivadas dos outros metamodelos, assim na implementação em banco de dados relacional as informações são extraídas através de *views*. Devido ao tamanho dos códigos SQL para a geração das Views, o código está no Anexo A.

Foram apresentadas algumas pesquisas na seção 3.3.1. Essas pesquisas auxiliam o modelador a responder perguntas apresentadas ao modelador baseado no histórico.

O estereótipo mais usual para um conceito é recuperado através do código:

```
SQL:
select count(1), stereotype_id from view.modelelement_vw me
left outer join view.model_element_stereotype_vw mes on me.id = mes.element_id
left outer join view.stereotype_vw st on mes.stereotype_id = st.id
where
lower(me.NAME) = lower('TYPE.NAME')
group by mes.stereotype_id
```

Essa pesquisa em SQL recuperará o número de instâncias estereotipadas com um determinado estereótipo. A ferramenta posteriormente calcula a percentagem de cada estereótipo e apresenta para o usuário.

O estereótipo mais usual para um tipo que desempenha um determinado papel:

```
SQL:
select count(1), stereotype_id from view.modelelement_vw me
left outer join view.model_element_stereotype_vw mes on me.id = mes.element_id
left outer join view.stereotype_vw st on mes.stereotype_id = st.id
left outer join view.classification_vw c on c.element_id = me.id
where
lower(me.NAME) = lower('TYPE.NAME')  and
c.role_id = ROLE_ID
group by mes.stereotype_id
```

Onde TYPE.NAME é o nome do tipo e ROLE_ID é um código para o papel desempenhado pelo tipo. A descrição dos códigos estão na tabela view.role.

A Pesquisa abaixo recupera todos os elementos de uma partição *Phase,* dado uma fase do tipo, cujo nome é TYPE.NAME e o nome do supertipo é TYPE.SUPER.NAME. A pesquisa é ordenada por número de vezes que os nomes são utilizados para os tipos participantes da partição.

```
SQL:
select lower(mtarget.name), count(1) from view.classification_association_vw  c
  left outer join view.modelelement_vw mtarget on mtarget.id = c.target_id
  left outer join view.modelelement_vw msource on msource.id = c.source_id
  where
  c.association_type_id = 11 and
  msource.name = ' TYPE.NAME '
  and msource.id in ( select mtarget.id from view.classification_association_vw  c
  left outer join view.modelelement_vw mtarget on mtarget.id = c.target_id
  left outer join view.modelelement_vw msource on msource.id = c.source_id
  where
  c.association_type_id = 10 and
  msource.name = ' TYPE.SUPER.NAME ')
  group by mtarget.name
  order by count(1)
```

Pequenas modificações são necessárias na pesquisa para que ela retorne os nomes mais utilizados para os tipos participantes em uma partição SubKind.

```
SQL:
select lower(mtarget.name), count(1) from view.classification_association_vw  c
  left outer join view.modelelement_vw mtarget on mtarget.id = c.target_id
  left outer join view.modelelement_vw msource on msource.id = c.source_id
  where
  c.association_type_id = 21 and
  msource.name = ' TYPE.NAME '
```

```
  and msource.id in ( select mtarget.id from view.classification_association_vw  c
  left outer join view.modelelement_vw mtarget on mtarget.id = c.target_id
  left outer join view.modelelement_vw msource on msource.id = c.source_id
  where
  c.association_type_id = 20 and
  msource.name = ' TYPE.SUPER.NAME ')
  group by mtarget.name
  order by count(1)
```

Outra sugestão que é possível recuperar do modelo é com qual conceito um tipo estereotipado como *Role* tem a relação de dependência de classe. Neste caso é requerido recuperar duas informações. Primeiro a pesquisa para recuperar o nome do conceito.

```
SQL:
select me.name,  count(1)  from  view.modelelement_vw  me
  left outer join view.model_element_stereotype_vw mes on me.id = mes.element_id
  left outer join view.stereotype_vw st on mes.stereotype_id = st.id
  where
  me.id in (
select distinct me.id from view.modelelement_vw  me
  left outer join view.classification_vw c on me.id = c.element_id
  left outer join view.classification_association_vw ca on c.element_id = ca.target_id
  left outer join view.modelelement_vw  meSource on ca.source_id = meSource.id
  where
ca.association_type_id = 33 and
meSource.name='TYPE.NAME'
)
group by me.name
```

A segunda pesquisa é para descobrir qual o nome da relação de dependência.

```
SQL:
select me.name,  count(1)  from  view.modelelement_vw  me
  left outer join view.model_element_stereotype_vw mes on me.id = mes.element_id
  left outer join view.stereotype_vw st on mes.stereotype_id = st.id
  where
  me.id in (
select distinct me.id from view.modelelement_vw  me
  left outer join view.classification_vw c on me.id = c.element_id
  left outer join view.classification_association_vw ca on c.element_id = ca.source_id
  left outer join view.modelelement_vw  meSource on ca.target_id = meSource.id
  where
ca.association_type_id = 32 and
lower(meSource.name)=lower('TYPE.NAME')
)
group by me.name
```

O DPModel, além de servir como base histórica para a criação de sugestões, também captura o *rationale* do modelo por meio das interações da ferramenta com o modelador. Cada interação do assistente apresentado na seção anterior com o modelador é enviada ao servidor e registrada no DPModel.

Por exemplo, quando o modelador responde a pergunta acerca da rigidez do conceito Pessoa, é criada uma instância da classe *LexicalPatternVerbalization* que registra o valor da metapropriedade, no caso falso. O atributo texto contém a frase "Não é possível que uma instância que não é uma Pessoa se torne uma Pessoa, ou que um individuo que é uma Pessoa deixe de ser uma Pessoa".

Instâncias dessa classe também são geradas para cada frase escolhida pelo sistema de padrões textuais que reflitam o domínio modelado. Portanto, caso o modelador queira compreender o *rationale* da estereotipação de um conceito no modelo, é possível consultar o DPModel.

5.3. SUBSISTEMA DE PADRÕES TEXTUAIS

A ferramenta implementa, além do guia metodológico e da sugestão por histórico, um sistema de auxílio ao modelador por padrões textuais, como apresentado na seção 4.2.1. Este sistema, devido às propriedades da linguagem natural discutidas anteriormente – ambígua e não lúcida – não apresenta sempre as respostas corretas, mas é uma importante fonte de auxílio ao modelador como será demonstrado. Nesta seção, primeiramente será demonstrado o processo de captura das frases, depois o processo de filtragem dessas frases e, por último, a utilização delas pela ferramenta.

O subsistema de padrões textuais primeiramente necessita um grande corpo de texto para funcionar, pois a linguagem OntoUML pode ser utilizada para desenvolver modelos conceituais para qualquer domínio. Esse grande corpo de texto necessário é conseguido na Web, através da API[6] do buscador Google.

Na maioria das ferramentas de buscas, é possível pesquisar uma frase inteira ao colocarmos a frase entre aspas duplas. O subsistema, aqui proposto, tira proveito deste fato montando pesquisas por padrões textuais que justifiquem o valor de uma metapropriedade.

A Figura 29 demonstra o processo completo de escolhas de frases, o primeiro passo é escolher os padrões textuais utilizados de acordo com a metapropriedade a ser

[6] Mais informações acerca da API de pesquisa do Google podem ser encontradas nesta URL: http://code.google.com/intl/pt-BR/apis/ajaxsearch/

pesquisada. Este catálogo de padrões textuais é armazenado no formato XML. É possível visualizar esse arquivo completo no Anexo B.

Figura 29 – Subsistema de Padrões Textuais

Para armazenar as informações de pesquisa, foi definido um arquivo XML – trecho na Figura 30 - onde são armazenados: a metapropriedade a que o padrão diz respeito; qual o valor da metapropriedade que ele evidencia; a variável – marcação <variable> – que será a sequência de caracteres substituída no momento da pesquisa pelo nome do conceito; a *query* é o padrão que será pesquisado, por exemplo, para o conceito "*Actor*" a pesquisa feita será: "*became an Actor*".

```
<?xml version="1.0" encoding="UTF-8"?>
<patterns>
    <pattern meta-property="rigidity">
        <variable>-x-</variable>
        <corpus evidence="false" linguistic-filter="true">
          <regularexpressions>
              <expression>\bbecame\b.*?(a|an).*?(NN|NP|NNS|NPS)\t\b-x-\b\s[^?(NN|NS|NPS|NNS)]</expression>
          </regularexpressions>
          <queries>
              <query>became a -x-</query>
              <query>became an -x-</query>
          </queries>
        </corpus>
    </pattern>
```

Figura 30 - Arquivo XML com os padrões textuais pesquisados

Após a submissão da *query,*e a pesquisa no Google nos retorna pequenos trechos como resultados – Figura 31 – conhecidos como *snippets*. Esses pequenos trechos são utilizados para fazer as pesquisas por padrões textuais.

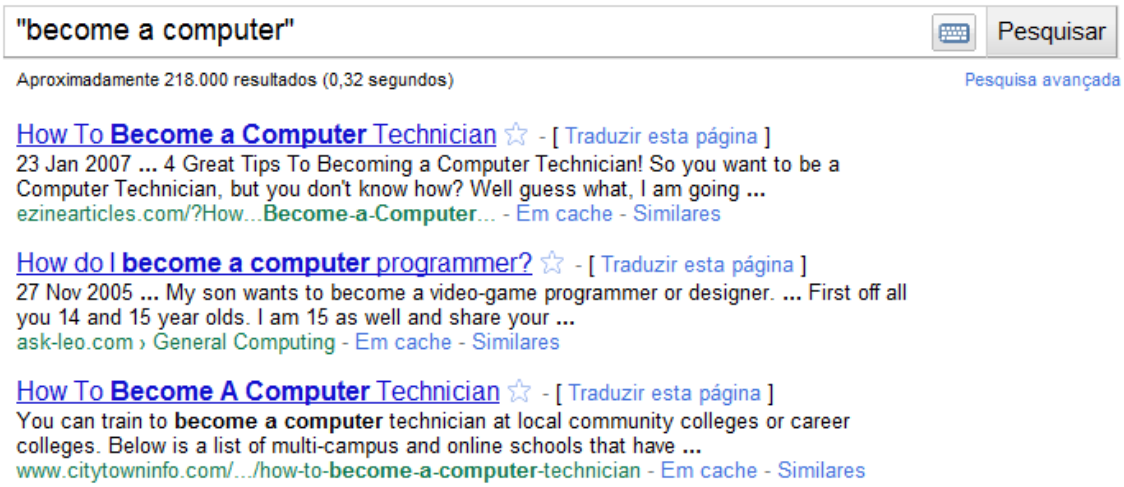

Figura 31 - Resultados contendo trechos, conhecidos como snippets.

Os resultados conseguidos passam por um último filtro, com o objetivo de eliminar resultados indesejados. Por exemplo, ao se pesquisar o padrão "became a *Computer"* a seguinte frase poderia ser retornada como resultado da busca:

> *"...became a Computer Technician..."*

Esta frase claramente não diz respeito ao termo "*Computer*" e sim ao termo "*Computer Technician*". Para tratar esses casos, foi utilizado um *part-of-speech* tagger (POS Tagger). Em (VÖLKER, VRANDECIC, *et al.*, 2008), foi demonstrado que a utilização deste tagger é muito eficiente para eliminar evidências falsas em padrões textuais. Um POS Tagger coloca marcas nas palavras classificando-as de acordo com a sua função gramatical usando informação contextual (SCHMID, 1994). No caso exemplificado pelo termo "*Computer",* seriam aceitos como evidências apenas textos que não contivessem outro substantivo após *computer* evitando construções como: *computer hacker, computer problem.*

A ferramenta de POS Tagger não filtra propriamente o resultado de texto, ela apenas marca os resultados. O resultado marcado passa, então, por um filtro de expressões regulares que realizam o processo de filtragem propriamente dito. As expressões regulares pesquisadas são encontradas no arquivo XML, na marcação *expression*. As expressões pesquisadas incluem a marcação feita pelo POS Tagger. Este filtro mostrou-se bastante útil, pois torna possível diferenciar casos como o demonstrado evitando, por exemplo, que a frase *"became a Computer Technician"* evidencie, neste caso, a não rigidez ou qualquer outra metapropriedade do termo "*Computer"*. De maneira geral, o

uso do POS tagger, portanto, ajuda a prevenir a falsa atribuição de metapropriedades pelo subsistema de busca por padrões léxicos.

Após o processo de seleção das frases, essas são apresentadas ao modelador – Figura 32 – quando este encontrar frases que correspondam ao seu modelo de maneira que possam ser utilizadas como justificativa para a escolha do modelador.

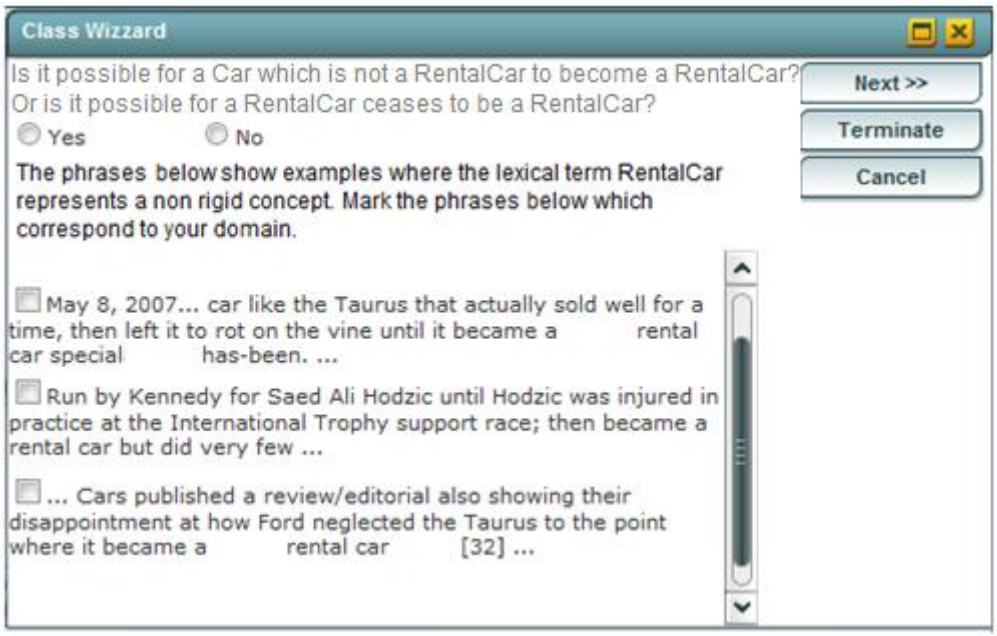

Figura 32 - Exemplo de apresentação de frases

5.4. ESTUDO DE CASO

De posse do editor, como um estudo de caso, foi criado na ferramenta um modelo de exemplo – Figura 33 – baseado no modelo presente na seção 8.4 de (GUIZZARDI, 2005), que mostra a união de várias ontologias de domínio. Os elementos do modelo são apresentados com os respectivos estereótipos. No final será possível comparar o modelo resultante com o modelo original.

O Objetivo deste exemplo é demonstrar a criação de um modelo conceitual em OntoUML sem que o usuário tenha que estereotipar diretamente alguma classe. Desta maneira, o modelador não necessita conhecer a linguagem para utilizar esta aplicação. Se, ao final do caso de uso, o modelo obtido for igual ao modelo original, pode-se considerar que o funcionamento foi bem sucedido na criação desse modelo.

Para a demonstração da utilização também foram criados outros modelos na ferramenta sendo esses exemplos retirados de artigos publicados para que a camada de sugestão baseadas em histórico tenha dados para inferir resultados., alguns destes modelos estão inclusos no Anexo C.

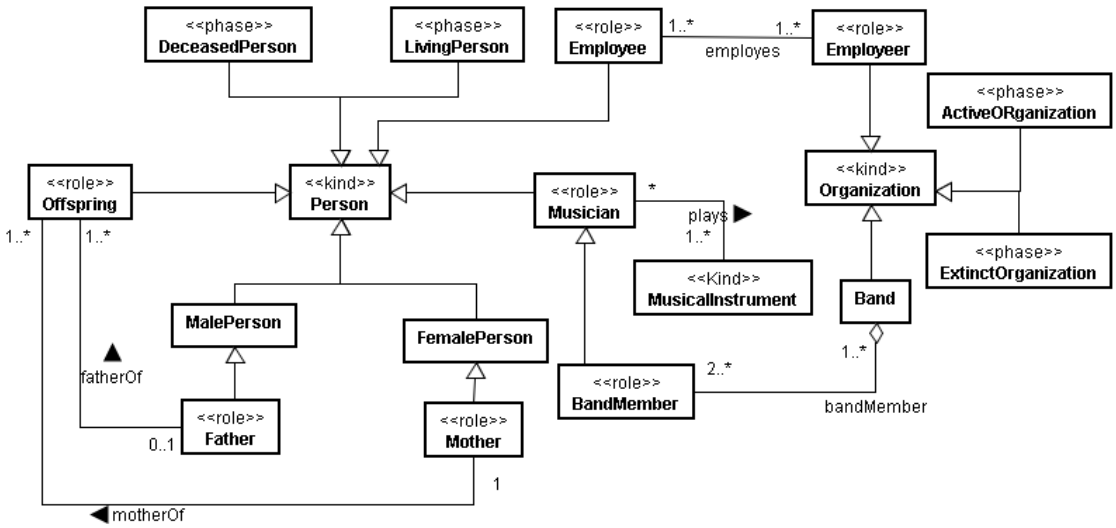

Figura 33 – Exemplo baseado em modelo presente na seção 8.4 de (GUIZZARDI, 2005)

Para utilizar a ferramenta, é necessário registrar-se com um usuário e senha. Esses dados são utilizados para atribuir o proprietário do modelo, e o autor das estereotipações. Após o processo de registro, abre-se a ferramenta de modelagem e atribui-se um nome e uma descrição – Figura 34 – para o modelo para que este possa ser identificado na ferramenta posteriormente. Este diálogo pode ser acessado no item de menu *File->Edit Model Information.*

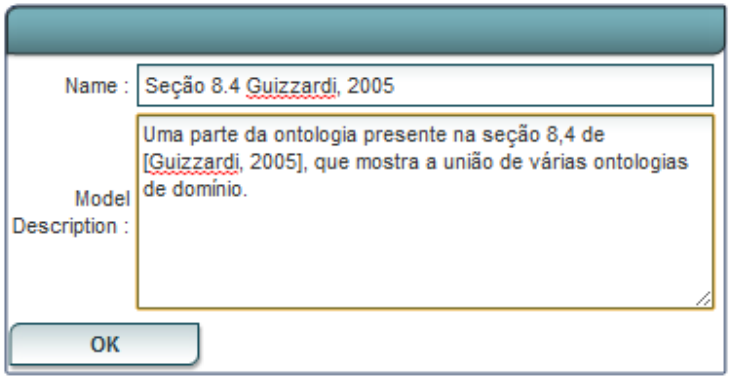

Figura 34 - Descrição do Modelo Criado

Após ser criada a identificação inicia-se o processo de criação do modelo. Clique no construtor *Class* e crie uma classe no modelo. Logo aparecerá um diálogo, demonstrado na Figura 35, que pergunta o nome do conceito.

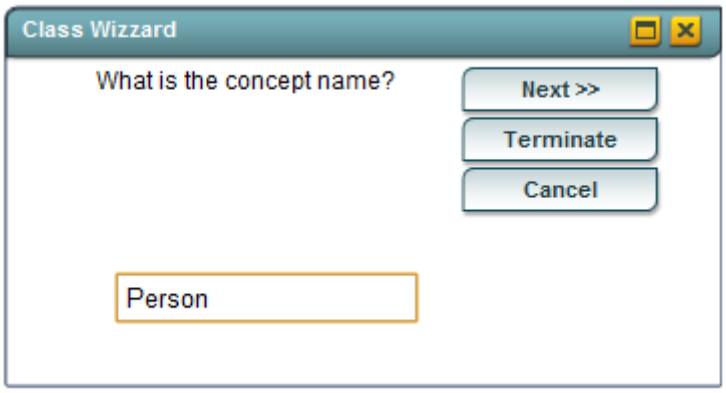

Figura 35 - Diálogo que pergunta o nome da classe criada

Após ser respondida esta pergunta, aparece um novo diálogo acerca da hierarquia da classe, representado na Figura 36. Como o conceito *Person* não é subtipo de nenhum outro conceito a resposta é *"Other type does not subsumes Person"*.

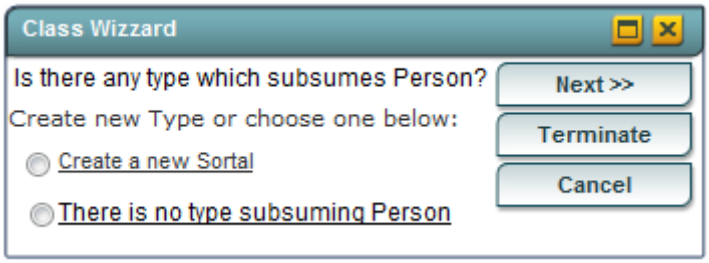

Figura 36 - Escolha de Supertipos

O terceiro diálogo – Figura 37 – refere-se à metapropriedade rigidez do conceito *Person.* A pergunta é se existe a possibilidade de um individuo que é uma pessoa deixar de ser uma pessoa em outro momento. Este fato como discutido demonstra a não rigidez do conceito. Este não é o caso do conceito *Person*, logo a resposta é "não". No diálogo também aparece algumas frases, escolhidas pelo sistema de padrões textuais, que exemplificam domínios onde é possível a não rigidez do conceito *Person* nenhum deles se encaixa com o caso apresentado. Quando o modelador responde "Não" e registra esta informação na ferramenta é criada uma instância no DPModel de *LexicalPatternVerbalization* com o texto "Não é possível que um individuo que não seja uma Pessoa se torne uma Pessoa e também não é possível que uma Pessoa cesse de ser uma Pessoa". Todas as interações do usuário com os diálogos do assistente acerca das metapropriedades geram essas instâncias de *LexicalPatternVerbalization,* consequentemente registrando esta informação no DPModel.

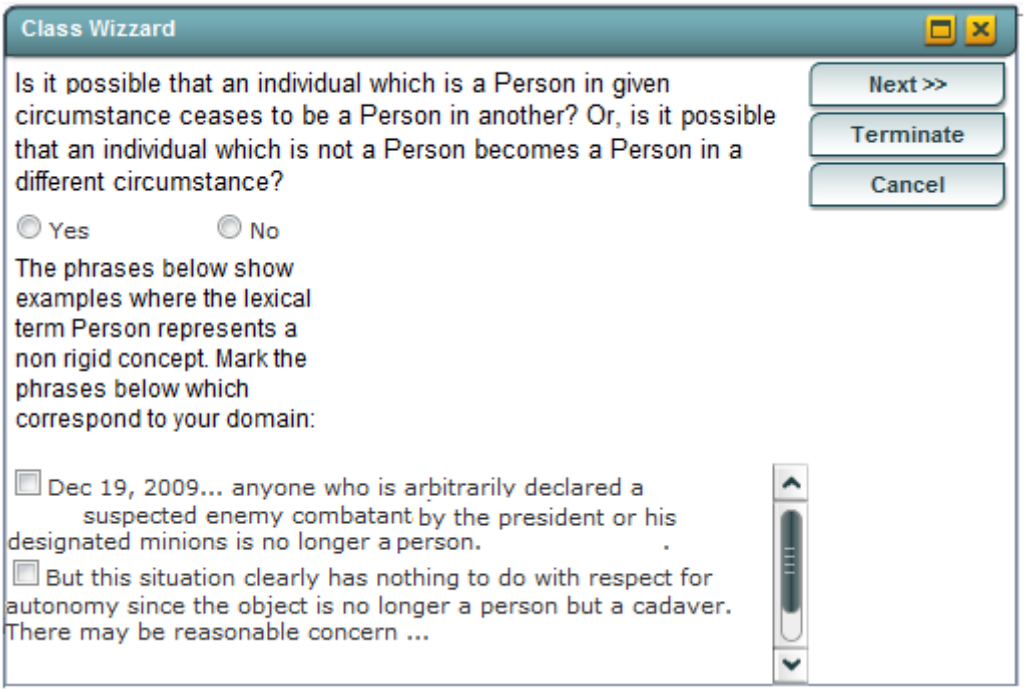

Figura 37 - Diálogo acerca da Rigidez do conceito *Person*

O quarto diálogo diz respeito se o conceito provê algum princípio de identidade. No nosso domínio este fato é verdadeiro *Person* provê o princípio de identidade.

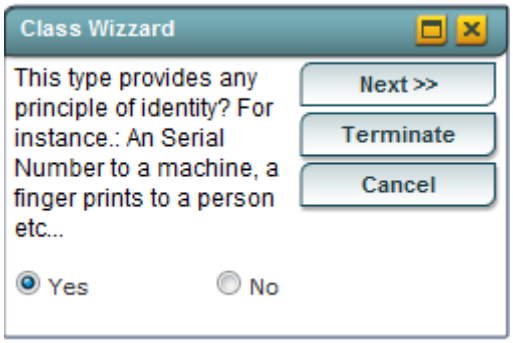

Figura 38 - Diálogo acerca do princípio de Identidade

Com estas informações a ferramenta está apta a inferir que o estereótipo do tipo *Person* é *Kind*. O próximo conceito a ser inserido é o *MalePerson*, subtipo do tipo *Person,* informação inserida no respectivo diálogo representado na Figura 39.

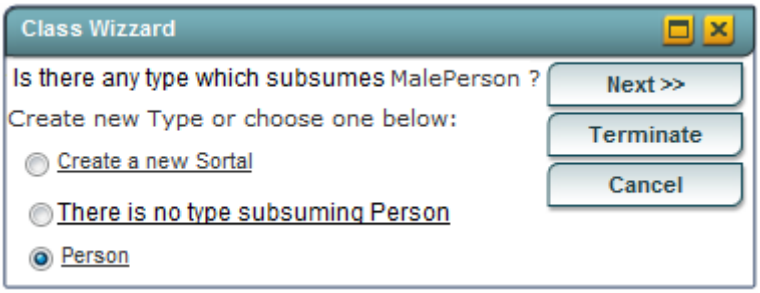

Figura 39 - Diálogo acerca da Hierarquia de *MalePerson*

No nosso domínio *MalePerson* será um conceito rígido, é possível visualizar na Figura 39 que o diálogo faz uso da hierarquia para formular a pergunta acerca da rigidez.

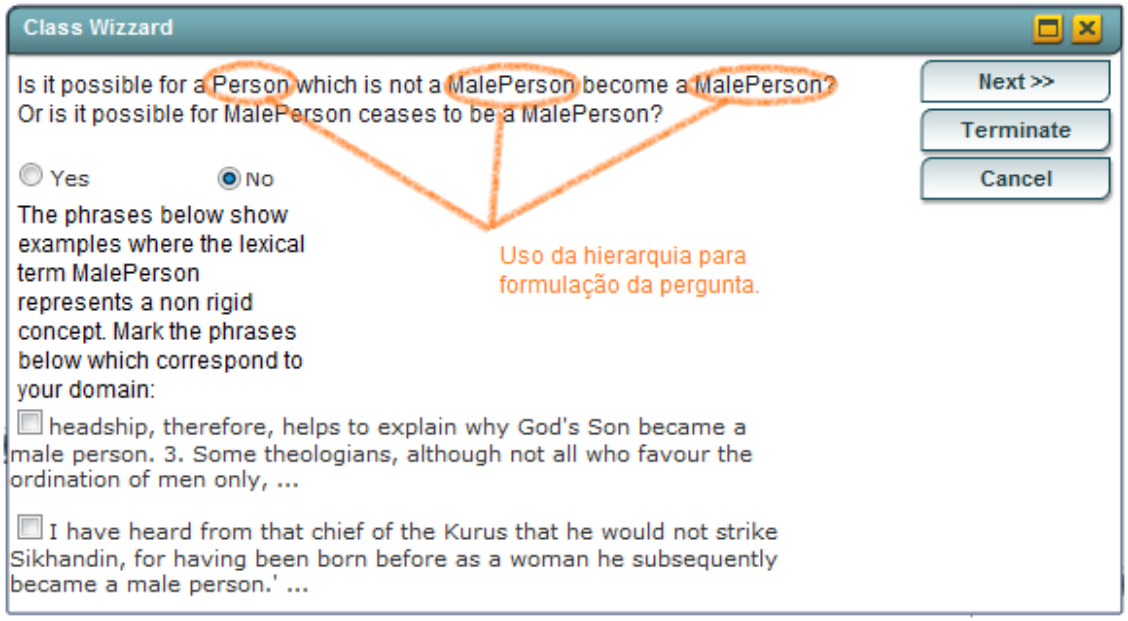

Figura 40 - Diálogo acerca da Metapropriedade rigidez do conceito *MalePerson* demonstrando a utilização da hierarquia para formulação da frase.

Utilizando a informação presente no modelo de que o princípio de identidade é provido pelo conceito *Person* ele infere que o estereótipo de *MalePerson* é *SubKind.* Como demonstrado na regra R_{SK} esse estereótipo pode aparecer como uma partição logo o próximo diálogo representado na Figura 41. A partir da resposta inserida no diálogo, conforme a regra R_{SK} já é criado outra classe *FemalePerson* estereotipada como *SubKind* e subtipo de *Person.*

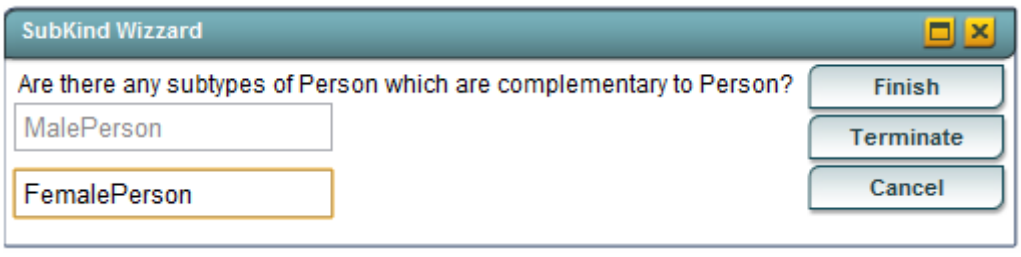

Figura 41 - Partição SubKind

O próximo conceito a ser modelado é o conceito *Father* subtipo de *MalePerson* este conceito é antirrígido como demonstrado no diálogo representado na Figura 42. É possível visualizar neste diálogo também que as frases escolhidas pelo sistema de

padrões textuais podem justificar a escolha, exceto a terceira fase por ser uma abordagem emocional do Pai que não espelha o domínio modelado.

Estas frases são adicionadas ao DPModel como justificativa a estereotipaçao do conceito Father. Cada frase selecionada é instanciada como um tipo de LeixcalPatternVerbalization relacionada com o a metapropriedade rigidez. Estando disponível para outros modeladores utilizarem esta informação posteriormente.

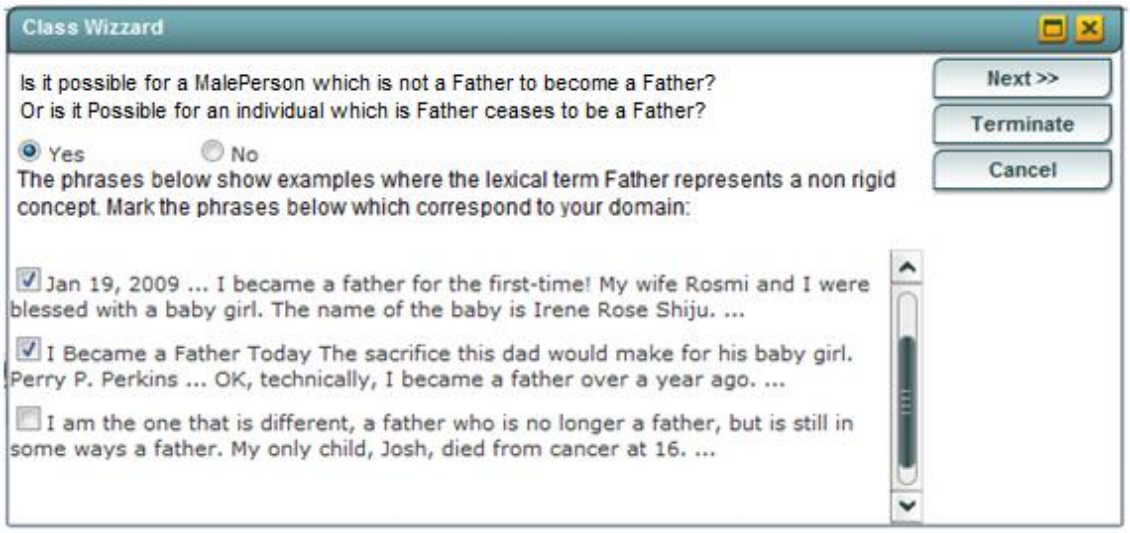

Figura 42 – Diálogo acerca da metaproproriedade do conceito Father

O conceito *Father* é antirrígido, logo pelo fluxograma apresentado na Figura 23 da seção 4.1 a próxima pergunta será acerca da dependência relacional deste conceito, é possível visualizar o diálogo na Figura 43. Um homem para ser Pai tem que ter um relacionamento de paternidade com outra pessoa, logo é um conceito relacionalmente dependente, isto pode ser demonstrado nos exemplos presentes no diálogo. As frases marcadas são adicionadas ao DPModel como registrado anteriormente.

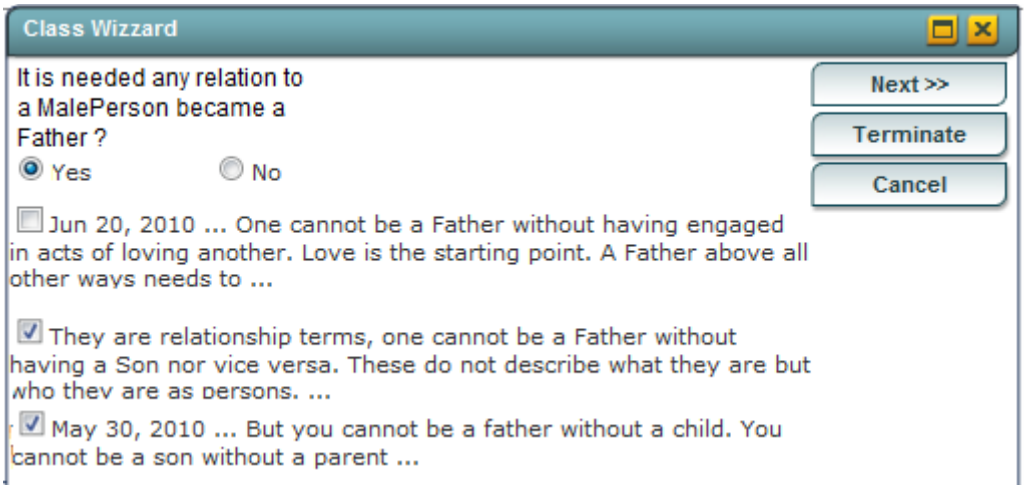

Figura 43 - Diálogo acerca da dependência relacional

Com as informações capturadas acerca do conceito é possível inferir que pode ser estereotipado como *Role*. Pelas regras R_R pode-se perguntar ao modelador com qual entidade o conceito é relacionado e qual é o nome deste relacionamento, é possível visualizar este diálogo na Figura 44. O conceito *Father* é relacionado com o conceito *Offspring* pelo relacionamento *fatherOf*.

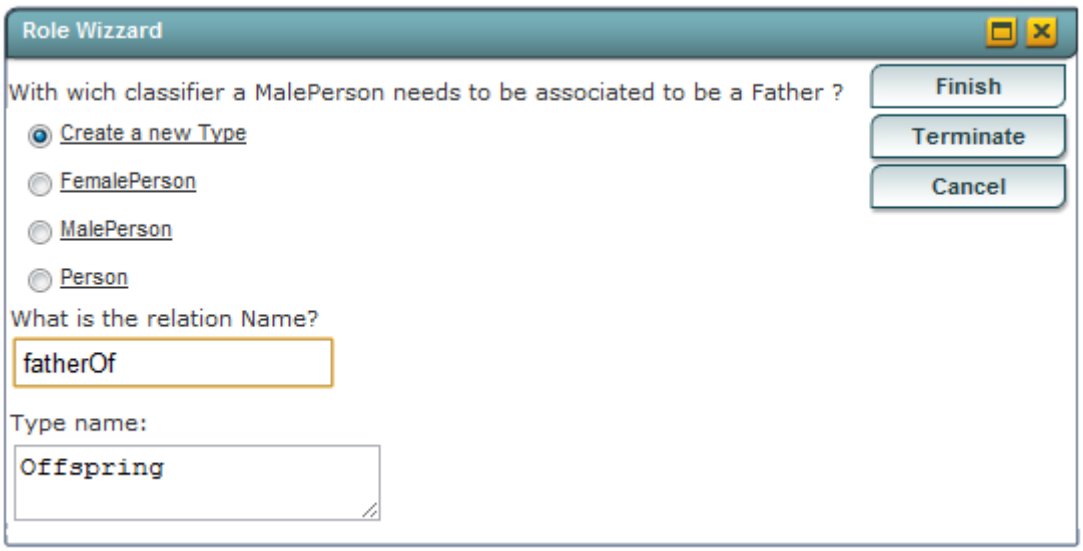

Figura 44 - Diálogo acerca das relações do conceito *Father*

A inserção do conceito *Mother* subtipo de *FemalePerson* segue os mesmos passos, com o relacionamento entre *Offspring* e *Mother* com o nome de *motherOf*.

O próximo conceito a ser inserido são os conceitos *LivingPerson*. Os diálogos seguem os mesmos passos especificados para as outras entidades o nome do tipo é *LivingPerson*

e é um subtipo de pessoa, uma *LivingPerson* pode mudar o seu estado e não é necessário nenhum relacionamento com uma entidade externa para um indivíduo do conceito *Person* ser um *LivingPerson*. Logo este conceito é estereotipado como uma fase. Desta maneira aparece um diálogo – Figura 45 – para extrair os tipos complementares dessa fase, dado que uma fase sempre aparece em partições.

É possível visualizar que o diálogo contém uma sugestão. Esta sugestão é fornecida pelo DPModel com a pesquisa que recupera os mais comuns de uma partição *Phase* dado o elemento e o supertipo. É possível visualizar o SQL dessa pesquisa na seção 5.2. O modelo resultante após essas inserções está na Figura 46.

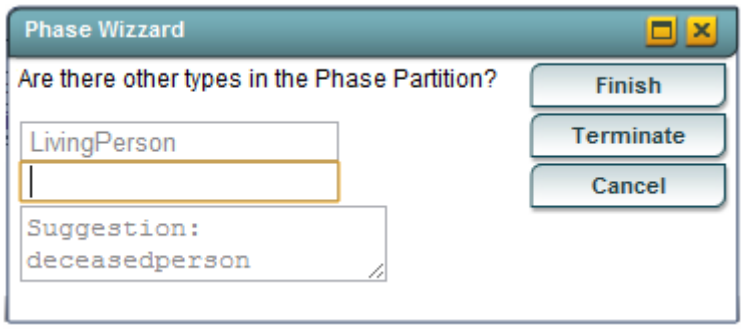

Figura 45 - Diálogo acerca das fases complementares

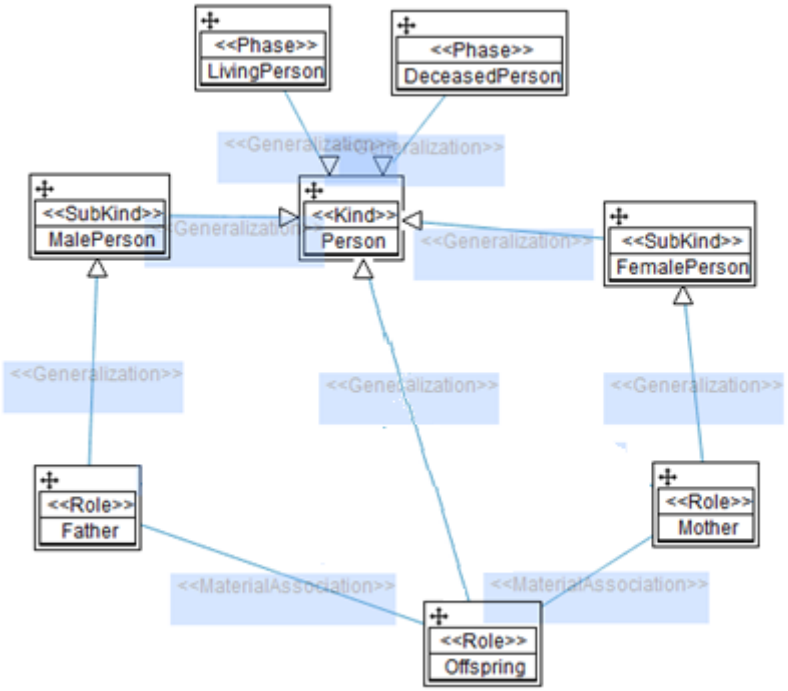

Figura 46 - Modelo representado na Ferramenta

O próximo conceito a ser inserido é Empregado (*Employee)*, os diálogos disparados pela inserção deste conceito no modelo são: 1)Empregado é subtipo de algum tipo: Pessoa(*Person).* Desta informação a ferramenta infere que o principio de identidade é herdade do conceito Pessoa 2)Uma pessoa(*Person*) que não é um empregado(*Employee*) pode se tornar um empregado? Sim. 3)É necessário algum relacionamento para uma pessoa ser um empregado? Sim; Estes diálogos determinaram a estereotipação de Empregado como *Role*. 4)Qual o relacionamento? empregadoDe(*employedAt*); Com que entidade é necessário ter um relacionamento? Empregador, este conceito não existia no modelo e é criado um novo conceito para ele.

O diálogo referente a pergunta 4 está na Figura 47, é possível visualizar as sugestões providas pela ferramenta, com base nas informações históricas. É interessante visualizar a sugestões dadas pela aplicação. A aplicação surgere corretamente o relacionamento "*employedAt",* entretanto surgere Organização(*Organization*) para o relacionamento alvo – visualizar modelos no Anexo C – como é possível no domínio modelado é utilizado uma especialização de Organização, Empregador(*Employer*).

A informação acerca do princípio de identidade é inferida da hierarquia, como o conceito de identidade é provido por *Person* ele não provê este conceito.

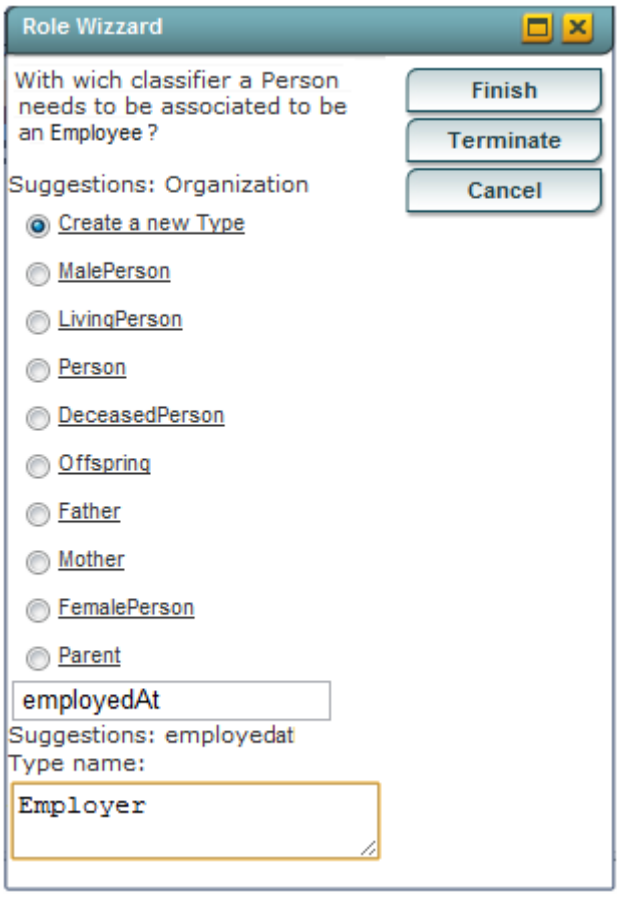

Figura 47 - Criação de Novo conceito Employer

Para o conceito criado Empregador (*Employer*) seguem os diálogos: 1) Empregador é um subtipo de algum conceito? Sim. Criar nova entidade Organização (*Organization*). 2)Uma organização que não é uma empregadora pode se tornar uma empregadora? Sim. Seguindo o fluxo proposto na Figura 23 segue a próxima pergunta 3) É necessário algum relacionamento para uma Organização se tornar uma empregadora? Sim. O conceito é estereotipado como Role e segue o próximo diálogo pela regra R_R. 4) Com que entidade é necessário ter um relacionamento? Empregado. Como já existe uma relação entre as duas entidades a regra termina.

A entidade organização tem os seguintes diálogos: 1) Organização é subtipo de outro conceito? Não; 2) Uma entidade que não é uma organização pode se tornar uma organização? Não; 3) Uma organização provê algum princípio de identidade? Sim. Com estas informações é possível estereotipar *Organization* como *Kind* e o assistente para este conceito é finalizado.

Será inserido agora os conceitos relativos ao assunto música no modelo. O primeiro será Músico(*Musician*). 1) Músico é subtipo de outro tipo? Sim. Pessoa; 2) Uma pessoa que não é um músico pode se tornar um músico? Sim; Pelo fluxo proposto na Figura 23 seguem as perguntas. 3) É necessário algum relacionamento para uma Pessoa se tornar um Músico? Sim; 4) Qual o relacionamento necessário? Tocar; Com que entidade é necessário ter um relacionamento? Criar entidade Instrumento Musical (*MusicalInstrument*).

Para entidade Instrumento Musical (*MusicalInstrument*) os diálogos são: 1) Instrumento Músical é subtipo de outro tipo? Não. 2) Uma entidade que não é um instrumento musical pode se tornar um instrumento musical? Não. Pelo fluxo proposto na Figura 23 seguem as perguntas acerca do principio de identidade. 3) Instrumento Musical provê algum princípio de identidade? Sim. Com estas informações é possível inferir que a entidade pode ser estereotipada como *Kind.*

O próximo conceito será Membro de Banda (*BandMember*). 1) Membro de Banda é subtipo de outro tipo? Sim. Músico 2) Um músico que não é membro de banda pode se tornar um membro de banda? Sim; 3) É necessário algum relacionamento para uma Músico se tornar um Membro de Banda? Sim; 4) Qual o relacionamento necessário? Ser membro de(*memberOf*); Com que entidade é necessário ter um relacionamento? Criar a entidade Banda(Band).

Para a entidade recém-criada Banda seguem os seguintes diálogos. 1) Banda é subtipo de outro tipo? Sim. Organização (*Organization*) 2) Uma organização que não é uma banda pode se tornar uma Banda? Não. Com estas informações já é possível inferir que banda pode ser estereotipada com *SubKind.*

O próximo tipo a ser inserido será o conceito Organização Ativa (*ActiveOrganization*), seguem os seguintes diálogos: Organização Ativa é subtipo de outro tipo? Sim. Organização (*Organization*) 2) Uma Organização que não é uma Organização Ativa pode se tornar uma Organização Ativa ou Uma Organização Ativa pode deixar de ser uma Organização Ativa? Sim. 3) É necessário algum relacionamento para uma Organização se tornar um Organização Ativa? Não;

Com estes dados o sistema infere que o tipo pode ser estereotipado como *Phase*. Logo segue o diálogo especifico do estereótipo: Quais as fases complementares? Organização Extinta (*ExtinctOrganization*).

Após inseridos todos os elementos do modelo é possível visualizar Figura 48 que obtivemos os mesmos estereótipos do modelo original sem a necessidade de estereotipar diretamente os conceitos e durante todo o processo o modelador foi auxiliado pela ferramenta com sugestões por histórico e frases selecionadas por padrões textuais para justificar a escolha de uma metapropriedade facilitando a criação do modelo.

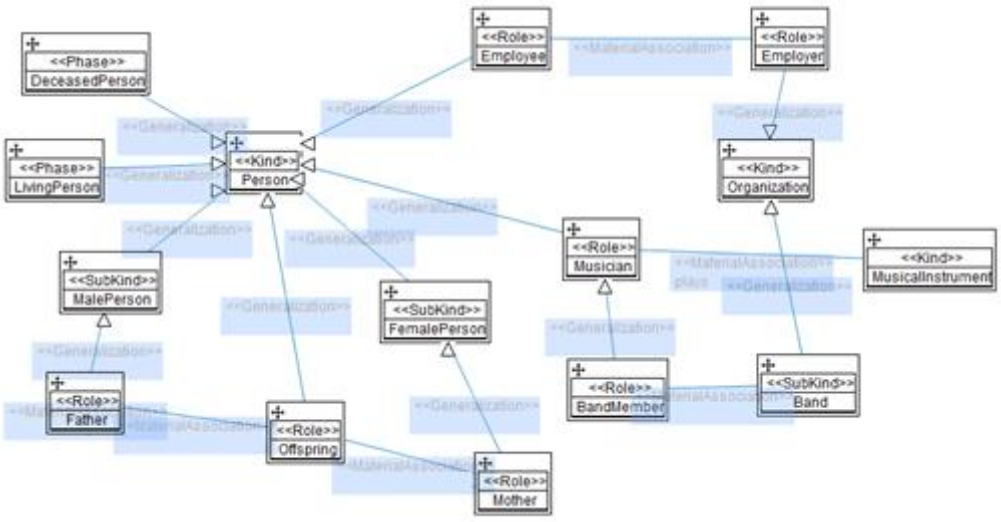

Figura 48 - Modelo Resultante

Outro ponto é que todas as interações da ferramenta com o modelador e o próprio modelo são depositadas no DPModel. Estas informações permitem primeiramente que outros modeladores obtenham o *rationale* do modelo, além de criar sugestões para outros domínios modelados, fortificando as sugestões da aplicação.

5.5. ARQUITETURA DO SISTEMA

Este editor é arquitetado de maneira a ser uma aplicação Web. Isto implica que a ferramenta está disponível para a utilização sem a necessidade de instalação, e é portável para qualquer sistema operacional com um navegador com suporte a JavaScript. Entretanto uma aplicação web também acarreta em algumas restrições na aplicação.

A principal restrição é que o código disponível no cliente não dispõe de todas as funcionalidades de um sistema *desktop*, como por exemplo: acesso ao banco de dados, acesso ao sistema de arquivo. Logo é necessário que o programa no cliente trabalhe em conjunto com o programa no servidor. Outra restrição é que a comunicação cliente servidor é feita através da Internet, o que pode levar a limitações de desempenho de acordo com a banda disponível no cliente.

Na Figura 49 é possível visualizar um diagrama do sistema. As camadas mais externas estão no cliente, as camadas internas estão no servidor e são acessadas através de chamadas remotas.

Figura 49 - Arquitetura do Sistema

É relevante afirmar que a ferramenta apresentada, durante o processo de construção teve como objetivo ser adaptável a outras linguagens. Deste modo é possível implementar a ferramenta para construir modelos em outras linguagens, desde que os metamodelos estejam codificados em ECore.

5.5.1. CAMADA CLIENTE

A camada cliente, executada no browser do usuário foi implementado utilizando a tecnologia AJAX(Asynchronous Javascript And XML) que torna aplicações WEB mais interativas, não sendo necessário recarregar a página para alterar o seu conteúdo. Para suportar a comunicação AJAX foi utilizada a biblioteca GWT (Google Web Toolkit) do Google. Além do GWT, foi utilizado o SmartGWT, uma biblioteca de componentes visuais para GWT.

A camada de cliente fica responsável por apresentar as sugestões e restrições para o cliente e tornar possível uma interação amigável. Entretanto, todo o processamento das sugestões e restrições dos modelos é feito no servidor.

A camada cliente não conhece completamente o metamodelo utilizado na ferramenta, tendo somente conhecimento, em tempo de execução, das metaclasses instanciáveis existentes no metamodelo e dos nomes delas. O processamento das restrições do metamodelo, a criação de sugestão e persistência é tratada no servidor.

5.5.2. CAMADA DE REPRESENTAÇÃO INDEPENDENTE DE METAMODELO

Esta camada é responsável por representar o metamodelo da linguagem e os seus metamodelos, pois não é possível a representação do metamodelo em ECore no cliente. Este fato é consequência do cliente não tem acesso à biblioteca EMF não sendo, portanto apto a processar as classes neste formato e muito menos processar as restrições em OCL.

Além disso, esta camada torna a ferramenta independente de metamodelo, de maneira que alterações no metamodelo não impliquem em alterações significativas na ferramenta. Para conseguir esta independência foram criadas classes capazes de representar tanto as metaclasses do metamodelo quanto as relações de instanciação. O diagrama deste pacote de classes está representado na Figura 50, este pacote será referenciado como CRIM(Camada independente de metamodelo) . Cada uma das classes é descrita a seguir:

- OntoEEntity: Classe abstrata que representa qualquer entidade no cliente. Como atributo contém um identificador único.
- OntoEMetaclasse: Representa as metaclasses que existem no metamodelo da linguagem utilizada para fazer modelagem.
    - OntoEClass: Representa Metaclasses que quando instanciadas serão Classificadores, ou seja não serão relações.
    - OntoERelation: Representa Metaclasses que Quando instanciadas serão representadas como relações entre dois classifiers.
- OntoEInstance: Representam elementos do modelo propriamente. O atributo modelID aponta para o modelo o qual é dono deste elemento de modelo.
    - OntoEAttribute: Represeta os atributos de um classificador. Um atributo deve ter um pai.
    - OntoEClassObject: Representa um classificador em um modelo.
    - OntoERelationObnject: Representa uma relação binária em um modelo.
- OntoERestriction: Representa valores de metapropriedades. Por exemplo, uma instância com a metapropriedade *isAbstract* verdadeira será relacionado com *OntoERestriction* [*name*=”*isAbstract”, type=”Boolean”, value=”true”*].

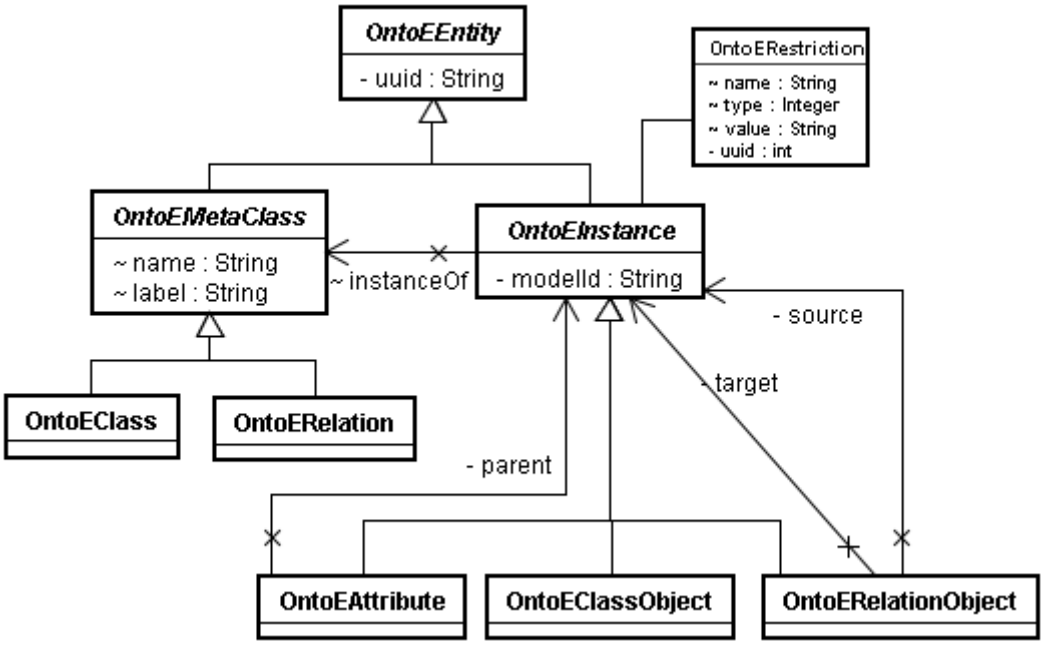

Figura 50 – Diagrama de Classes UML do pacote CRIM. Classes para representação dos metamodelos e modelos no cliente.

5.5.3. CAMADA DE MAPEAMENTO

A Camada de mapeamento é responsável por mapear a notação do pacote CRIM para a notação em *ECore* de um metamodelo específico. Este mapeamento é feito a cada ação do usuário que modifique o modelo. Esta camada também abriga outras duas funcionalidades altamente importantes: a camada de restrição; e a camada de sugestão.

A camada de mapeamento utiliza o conhecimento acerca do metamodelo para fazer a tradução entre os dois modelos. As metaclasses que são relacionamentos são mapeados como *OntoERelation*. As metaclasses que são Classificadores são mapeados como *OntoEClass*.

Como é demonstrado na Figura 51, todos os elementos subtipos de *Relationship* são mapeados como *OntoERelationObject*, todos os elementos subtipos de *Classifier*, exceto os *Datatype*,são mapeados como *OntoEClassObject*. E todos os elementos *Datatype* são mapeados como *OntoEAttribute*.

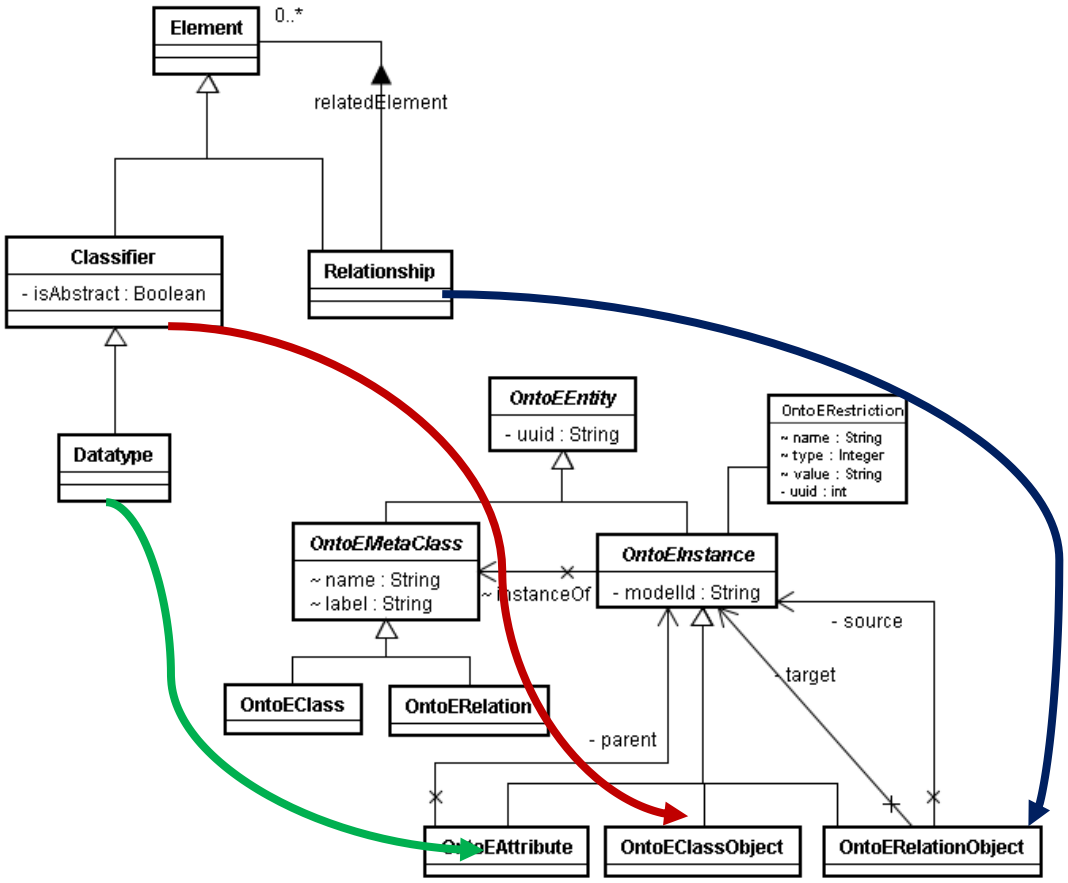

Figura 51 - Mapeamento OntoUML e DomainModel

O relacionamento *instanceOf* em *OntoEInstance*, relaciona uma instância a sua respectiva metaclasse. Por exemplo, um elemento em OntoUML instância da Metaclasse *Kind*, seria mapeado como um *OntoEClassObject* e a relação *instanceOf* apontaria para a instância de *OntoEClass* com o atributo *name* = "Kind".

A Classe *OntoERestriction* define o valor dos atributos das metaclasses, portanto, quando uma instância da metaclasse *Kind* tem o valor *isAbstract* = "true" ela teria um OntoERestriction [*name* = "*isAbstract*", *type*= "*Boolean*", *value* = "*true*"]

A camada de restrição implementa as restrições presentes na OntoUML, sendo responsável por evitar a criação de modelos sintaticamente incorretos. Foram utilizadas as restrições expressas na linguagem OCL como definido em (BENEVIDES, 2007). A checagem das restrições é feita sobre o modelo expresso em ECore, logo as restrições são checadas após o mapeamento de cada instância. Caso alguma regra seja quebrada

esta informação é enviada para o Cliente e podendo em certos casos a ação que levou o modelo a estar inconsistente ser desfeita.

A camada de sugestão implementa toda a metodologia de suporte ao modelador especificado neste trabalho e, portanto, será uma seção a parte.

5.5.4. CAMADA DE PERSISTÊNCIA

A persistência de dados é feita em um banco de dados relacional, utilizando a biblioteca ORM(Object Relathionship Mapping) Hibernate, para fazer o mapeamento do paradigma de Objetos para o paradigma relacional. Através dessa ferramenta o mapeamento é feito por meio de anotações JAVA nas classes. Facilitando o trabalho de criação de uma camada de Acesso a Dados. É possível visualizar na Figura 52 as tabelas criadas no banco de dados.

A persistência dos modelos é feita no formato do pacote CRIM, utilizado pelo cliente. Dessa maneira quando um modelo é carregado a camada de mapeamento é responsável por mapeá-la novamente para o formato ECore. Esta escolha também é balizada no requisito de que a ferramenta seja mantida o máximo possível independente de metamodelo, ou seja, a camada de persistência foi desenvolvida para suportar qualquer metamodelo ECore.

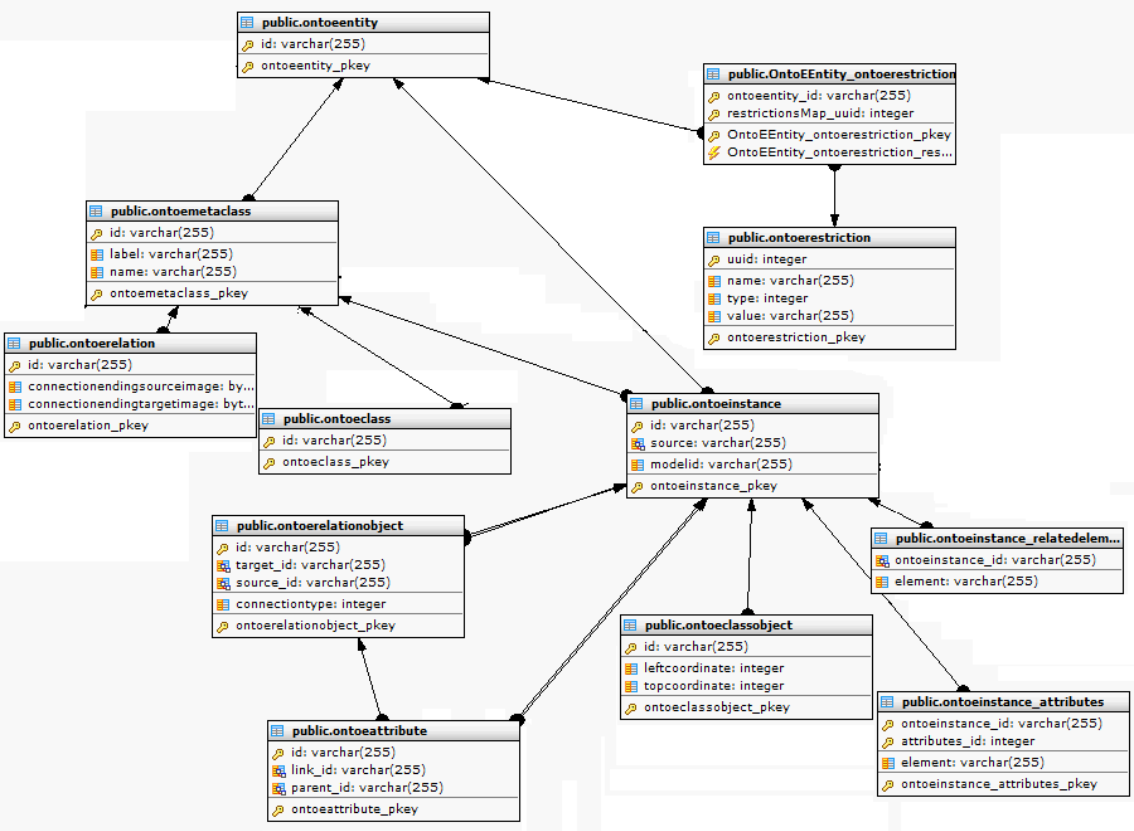

Figura 52 - Modelo do Banco de Dados do pacote de domínio

5.6. CONCLUSÃO

Este capítulo apresentou a ferramenta *OntoUML WebEditor*. A ferramenta por meio do das contribuições apresentadas neste trabalho auxilia a construção de modelos conceituais por modeladores inexperientes. Foram apresentados a implementação da ferramenta para: (i) o guia metodológico; (ii) suporte a estereotipação; (iii) sugestões históricas;(iv) padrões textuais.

A implementação da ferramenta, buscou sempre que possível, evitar interações desnecessárias com o modelador. Para isto foram utilizadas pesquisas OCL sobre o modelo ECore, e pesquisas sobre o modelo DPModel.

Também foi demonstrado que os padrões textuais utilizados além de apoiarem o modelador durante o processo de modelagem, capturam o design rationale do processo fortalecendo a base histórica do DPModel.

O caso de uso apresentado demonstrou a o processo de construção de um modelo na ferramenta. Durante o processo de modelagem as classes não foram estereotipadas diretamente, sempre beneficiando-se dos assistentes providos pela ferramenta para a estereotipação. Também foi possível visualizar o funcionamento do guia metodológico dando suporte ao modelador durante o processo de modelagem.

Neste capítulo também foi demonstrado as informações capturadas pelo DPModel. Foi demonstrado que estas informações são uteis para a construção de sugestões ao modelador. Tanto para o processo de estereotipação como para apresentar sugestões para diálogos no guia metodológico.

Por fim neste capítulo é apresentado a arquitetura da ferramenta, como frisado buscou-se na implementação da ferramenta construí-la de modo que fosse independente de metamodelo, dessa maneira a ferramenta é adaptável para outras linguagens com metamodelo em ECore. Entretanto para validar este fato seria necessário a adaptação da ferramenta para outra linguagem.

CAPÍTULO 6. CONSIDERAÇÕES FINAIS

Este capítulo apresenta as conclusões do trabalho realizado, mostrando suas contribuições. Descreve, ainda, possíveis trabalhos futuros que podem surgir baseados neste.

6.1. CONCLUSÕES

Este trabalho propôs métodos para apoiar o processo de criação de modelos ontologicamente bem fundamentados. Este tem sido um artefato que tem adquirido maior importância com o avanço da Web Semântica. O processo de criação de modelos conceituais que representem mais fielmente o domínio aos quais propõe representar é essencial para que os computadores consigam compreender melhor as informações.

Nada obstante, quão maiores e mais complexos os modelos maiores as dificuldades de integração entre eles, pois é natural que a consistência semântica dos modelos diminua com o crescimento destes. A utilização de uma linguagem de fundamentação se faz necessária para assegurar uma maior fidelidade à realidade (FIELDING, SIMON, *et al.*, 2004) e, por conseguinte, melhorar a integração dos modelos.

As contribuições deste trabalho objetivaram auxiliar o modelador no processo de criação de ontologias utilizando uma linguagem ontologicamente bem fundamentada. Entre as contribuições deste trabalho está o **desenvolvimento de um guia metodológico para auxiliar modeladores na construção de Ontologias** em OntoUML, uma linguagem ontologicamente bem fundamentada. Este guia utilizou os Padrões de Projetos presentes na linguagem para criar regras de formação do modelo e por meio delas auxiliar o modelador.

A segunda contribuição deste trabalho é um **guia para auxiliar a definição dos estereótipos dos conceitos em OntoUML**. É demonstrado um guia que auxilia o modelador na escolha do estereótipo de um conceito através de perguntas que não necessitem de conhecimento das definições ontológicas e também através de padrões textuais.

Outra contribuição deste trabalho é a criação do **DPModel, um modelo para capturar os padrões de projetos** presentes em um modelo, além das regras de derivação para

capturar os padrões de projetos em OntoUML. O DPModel é o modelo que torna capaz muitas das inferências feitas durante o desenvolvimento do trabalho, pois torna evidente as relações e os padrões de projetos dos modelos. Além disso, foi acoplado neste modelo conceitos para justificar cada estereotipação ocorrida na UFO. Esta informação pode ser utilizada para tornar o *rationale* dos modelos disponíveis para outros modeladores.

Também se demonstrou neste trabalho a utilização de **padrões léxicos** para capturar frases de textos que expressem um valor de uma metapropriedade. Estas frases são utilizadas para auxiliar o modelador no processo de estereotipação de um conceito. Além disso, quando um modelador assina-la que uma frase reflete o conceito no contexto do domínio modelado, esta frase é guardada como *rationale* do modelo.

Por último foi apresentado uma aplicação a *OntoUML WebEditor* que contém **uma implementação do guia metodológico, conjuntamente com todas as contribuições propostas** que tem por objetivo tornar menos complexo a construção de ontologias em OntoUML além de apoiar a modeladores inexperientes com dados históricos e padrões textuais.

Embora as propostas apresentadas facilitem o processo de criação de ontologias, é notável, que o trabalho foi desenvolvido utilizando apenas um fragmento da OntoUML, os tipos sortais da UFO-A, portanto, não está ainda apto a modelar qualquer domínio fielmente. Entretanto, apesar de não demonstrado neste trabalho, é possível estender esta proposta para todos os estereótipos da OntoUML. A utilização de toda a OntoUML, contendo classificadores e relacionamentos, traria vários outros padrões, fortalecendo as regras de formação apresentadas e também o guia de estereotipação, todavia, seria inviável abranger, em tempo hábil, todas os estereótipos somente nesta dissertação.

O guia para auxiliar a definição dos estereótipos e a utilização de padrões textuais para apresentar frases desenvolveu de uma primeira proposta, que seria a estereotipação automática através da detecção de padrões textuais em grandes corpos de textos. Entretanto, esta proposta demonstrou-se muito imprecisa devido à visão de mundo imprecisa expressa em muitos textos e às propriedades da linguagem natural que dificultam o seu processamento. Preferiu-se usar a ideia, porém, com interação com o modelador para garantir a escolha correta do estereótipo.

Este trabalho não contribui somente com as propostas apresentadas diretamente, mas também como um arcabouço para desenvolvimento de novas propostas para assistentes ao modelador e modelagem automática de modelos. O próprio desenvolvimento da aplicação desenvolvida neste trabalho considera este ponto, visto que a arquitetura da ferramenta é extensível para outras linguagens e a criação de novas regras e assistentes.

6.2. TRABALHOS FUTUROS

Este trabalho apresentou um guia metodológico para suporte ao modelador, contudo este guia apresentado neste trabalho aborda somente um fragmento da linguagem OntoUML. O primeiro passo é expandir este guia metodológico para toda a linguagem OntoUML, também é necessário consolidar a ferramenta, tratando problemas de interface com usuário para tornar o seu uso como ferramenta primária de modelagem possível.

Além disso, uma área de pesquisa seria a utilização das informações coletadas pelo DPModel além de sugestões para criação de modelos. Uma possibilidade seria a utilização dessa informação para auxiliar a compreensão automática de textos ou a geração de modelos através de textos. Quando um texto é lido não se conta somente com as informações contidas explicitas no seu conteúdo para compreendê-lo, tendo como suporte informações que são adquiridas durante a vida. O DPModel faria este papel de suporte apresentando relacionamento comuns entre as entidades.

Outro ponto que foi abordado neste trabalho e que pode ser aprofundado é a captura do *design rationale.* Durante o desenvolvimento desta dissertação foram feitas algumas pesquisas com a linguagem de representação de *design rationale* Kuaba(MEDEIROS, 2006) e existe a possibilidade nesta ferramenta de se aumentar a captura do *Design Rationale*, o que facilitaria o reuso das ontologias, além de possibilitar a melhor colaboração entre vários modeladores. Para completar essa proposta outro trabalho futuro seria a evolução da ferramenta para suportar construção de ontologias de maneira colaborativa, ou seja, simultaneamente por mais de um modelador.

BIBLIOGRAFIA

ACKOFF, R. L. From data to wisdom. **Journal of Applied Systems Analysis**, 1989.

ALDO GANGEMI, N. G. et al. **Sweetening Ontologies with DOLCE**. Proceedings of the 13th European Conference. Siguenza, Espanha: [s.n.]. 2002.

ANYANWU, K.; SHETH, A. **r-Queries:** Enabling Querying for Semantic Associations on the Semantic Web. 12th Intl. WWW Conference. Budapest, Hungria: [s.n.]. 2003.

AURNAGUE, L. V. A. M. Part-of Relations, Functionality and Dependence. **Categorization of Spatial Entities in Language and Cognition**, 2005.

BAUMAN, B. T. **Prying Apart Semantics and Implementation:** Generating XML Schemata directly from ontologically sound conceptual models. Balisage Markup Conference. [S.l.]: [s.n.]. 2009.

BENEVIDES, A. B. Titulo: **Uma Ferramenta baseada em Modelos para Modelagem Conceitual ontologicamente bem-fundada**. Dissertação (Mestrado em Informática) Vitória: [s.n.], 2007.

BERNERS-LEE, T. **Weaving the Web:** The Original Design and Ultimate Destiny of the World Wide Web. [S.l.]: Harper San Francisco, 1999.

BLOMQVIST, E.; ÖHGREN, A.; SANDKUHL, K. **ONTOLOGY CONSTRUCTION IN AN ENTERPRISE CONTEXT:** COMPARING AND EVALUATING TWO APPROACHES. [S.l.]: [s.n.], 2006.

BRAGA, B. F. B. et al. Transforming OntoUML into Alloy: towards conceptual model validation using a lightweight formal method, 2010.

BROWN, J. E. B. A. D. C. Rationale Support for Maintenance of Large Scale Systems, Massachusetts, 2003.

CIMIANO, P.; HANDSCHUH, S.; STAAB, S. **Towards the SelfAnnotating**. Proceedings of the 13th Word Wide Web Conference. New York: [s.n.]. 2004. p. 462-471.

CLANCEY, W. J. The Knowledge Level Reinterpreted: Modelling Socio-Technical Systems. **International Journal of Intelligent Systems**, 8, 1993. 33-49.

CORTADA, J. W.; WOODS, J. A. **The Knowledge Management Yearbook**. [S.l.]: Butterworth-Heinemann, 1999.

D'AQUIN, M. et al. **Ontology Modularization for Knowledge Selection:** Experiments and Evaluations. [S.l.]: [s.n.]. 2007.

DEVEDZIC, L. Understanding Ontological Engineering. [S.l.]: ACM, 2002.

DEVEDZIC, V. **Ontologies:** Borrowing From Software Patterns. Intelligence. New York: ACM. 1999.

ECLIPSE FOUNDATION. Model Development Tools. **Eclipse**, 2005. Disponivel em: <http://www.eclipse.org/modeling/mdt>. Acesso em: Agosto 2010.

ELENA PASLARU BONTAS, M. M. **Towards a reuse-oriented methodology for ontology engineering**. [S.l.]. 2005.

EUZENAT, J. **Knowledge Acquisition for Knowledge-Based Systems Workshop**. Proc. 10th Knowledge Acquisition for Knowledge-Based Systems Workshop. Banff: [s.n.]. 1996.

FERNÁNDEZ, M.; GÓMES-PÉREZ, A.; JUSTO, N. **METHONTOLOGY:** From Ontologycal Art Towards Ontological Engineering. AAAI97 Spring Symposium. Stanford: [s.n.]. 1997. p. 33-40.

FIELDING, J. M. et al. **Ontological Theory for Ontological Engineering:** Biomedical Systems Information Integration. Proceedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning. Whistler: [s.n.]. 2004.

FRECHIANI, F. **UMA ONTOLOGIA DE COLABORAÇÃO E SUAS APLICAÇÕES**. Dissertação (Mestrado em Informática) - Vitória, ES. 2009.

FRICKÉ, M. The knowledge pyramid: a critique of the DIKW hierarchy. **Journal of Information Science**, 2009.

GAMMA, E. et al. **Design Patterns:** Elements of Reusable Object-Oriented Software. [S.l.]: Addison-Wesley. 1995.

GANGEMI, A. **Ontology Design Patterns for Semantic Web Content**. [S.l.]: Springer, 2005.

GANGEMI, A. Design Patterns for Legal Ontology Contruction. In: _____ **Trends in Legal Knowledge:** The Semantic Web and the Regulation of Electronic Social Systems. [S.l.]: [s.n.], 2007.

GONÇALVES, B. N.; ZAMBORLINI, V.; GUIZZARDI, G. An Ontological Analysis of the Electrocardiogram. **RECIIS**, 3, 2009.

GOODMAN, H. S. L. A. N. The Calculus of Individuals and Its Uses. **J. Symbolic Logic**, 5, n. 2, 1940.

GRAÇAS, A. P. D. **Suporte Computacional Baseado em modelos para representação visual de Regras URML em modelos conceituais ontologicamente bem-fundamentados**. Vitória: [s.n.], 2008.

GRAÇAS, A.; GUIZZARDI, G. **Padrões de Modelagem e Regras de Construção de Modelos para a criação de Ontologias de Domínio Bem-Fundamentadas em OntoUML**. 3º SEMINÁRIO DE PESQUISA EM ONTOLOGIA NO BRASIL. Florianópolis, SC: [s.n.]. 2010.

GRUBER, T. et al. **Design Rationale Capture as Knowledge Acquisition Tradeoffs in the Design of Interactive Tools**. [S.l.]. 1991.

GRUBER, T. R. **Toward Principles for the Design of Ontologies Used for Knowledge Sharing**. Palo Alto: [s.n.], 1993.

GUARINO, N. The Ontological Level. **Philosophy and the Cognitive Science**, Vienna, p. 443-456, 1994.

GUARINO, N. Formal Ontology, Conceptual Analysis and Knowledge Representation. **INTERNATIONAL JOURNAL OF HUMAN AND COMPUTER STUDIES**, 43, 1995. 625-640.

GUARINO, N. **Understading, Building, and Using Ontologies**. International Journal of Human-Computer Studies, Vol. 46, No. 2-3. (February 1997), pp. 293-310. Padova, Italy: [s.n.]. 1997. p. 293-310.

GUARINO, N. **Formal Ontology and Information Systems**. [S.l.]: IOS Press, 1998.

GUARINO, N. **Formal ontology in information systems**. [S.l.]: IOS PRESS. 1998.

GUARINO, N. **Some Ontological Principles for Designing Upper Level Lexical Resources**. First International Conference on Language Resources and Evaluation. Granada, Spain: [s.n.]. 1998.

GUARINO, N.; WELTY, C. **A Formal Ontology of Properties**. Proceedings of the ECAI-2000 Workshop on Applications of Ontologies and Problem-Solving Methods. [S.l.]: [s.n.]. 2000.

GUIZZARDI, G. **Ontological Foundations for Structural Conceptual Models**. Enschede: [s.n.], 2005. ISBN 1381-3617.

GUIZZARDI, G. **ONTOLOGICAL FOUNDATIONS FOR STRUCTURAL CONCEPTUAL MODELS**. Enschede: [s.n.], 2005. ISBN 1381-3617.

GUIZZARDI, G. **The Role of Foundational Ontology for Conceptual Modeling and Domain Ontology Representation**. 7th International Baltic Conference on Databases and Information Systems. Vilnius, Lithuania: [s.n.]. 2006.

GUIZZARDI, G. **Modal Aspects of Object Types and Part-Whole Relations and the de re/de dicto distinction**. 19th International Conference on Advanced Information Systems Engineering (CAISE'07). Trondheim: Springer-Verlag. 2007.

GUIZZARDI, G. et al. **An Ontologically Well-Founded Profile for UML Conceptual Models**. 16th International Conference on Advances in Information Systems Engineering (CAiSE). Berlin: [s.n.]. 2004.

GUIZZARDI, G. et al. On the importance of truly ontological representation languages. **International Journal of Information Systems Modeling and Design (IJISMD)**, Hershey-New York, 1, 2010.

GUIZZARDI, G.; FALBO, R. D. A.; FILHO, J. G. P. Using objects and patterns to implement domain ontologies. **Journal of the Brazilian Computer Society**, Julho 2002. 44-56.

GUIZZARDI, G.; HALPIN, T. **Ontological Foundations for Conceptual Modeling, Applied Ontology**. [S.l.]: [s.n.], 2008. ISBN ISSN 1570-5838.

GUIZZARDI, G.; PIRES, L. F.; SINDEREN, M. V. **Ontology-Based Evaluation and Design of Domain-Specific Visual Modeling Languages**. Proceedings of the 14th

International Conference on Information Systems Development. Karlstad, Sweden: [s.n.]. 2005.

GUIZZARDI, G.; WAGNER, G. **A Unified Foundational Ontology and some Applications of it in Business Modeling**. [S.l.]: [s.n.], 2004.

GUIZZARDI, G.; WAGNER, G. **Towards Ontological Foundations for Agent Modeling Concepts using UFO**. Sixth International Bi-Conference Workshop on Agent-Oriented Information Systems. [S.l.]: Springer-Verlag. 2005.

GUIZZARDI, G.; WAGNER, G. **Towards and Ontological Foundation of Discrete Event Simulation**. 16th International Winter Simulation Conference. Baltimore, USA, : [s.n.]. 2010.

GURR, C. A. Effective Diagrammatic Communication: Syntatic, Semantic and Pragmatic Issues. **Journal of Visual Languages and Computing**, v. 10, p. 317-342, 1999.

HALPIN, T.; MORGAN, T. **Information Modeling and Relational Dababases**. ISBN 1558606726. ed. [S.l.]: Morgan Kaufman, 2008.

HEARST, M. A. **Automatic Acquisition of Hyponyms from Large Text Corpora**. Berkeley, California: [s.n.]. 1992.

HERRE, H. et al. **General formal ontology (GFO) – part I:** Basic principles. Onto-Med. [S.l.]: [s.n.]. 2006.

IEEE. **IEEE Standard Glossary of Computer Languages**. [S.l.]. 1993.

KLEIN, M. Capturing Design Rationale in Concurrent Engineering Teams. **Computer**, 26, n. 1, 1993.

KUNZ, W.; RITTEL, H. W. J. **ISSUES AS ELEMENTS OF INFORMATION SYSTEMS**. [S.l.]: [s.n.], 1970.

LEE, J.; LAI, K.-Y. **What's in design rationale?** [S.l.]: L. Erlbaum Associates Inc., 1996. 21 - 51 p.

LESNIEWSKI, S. Foundations of the General Theory of Sets. I, 1992.

LISKOV, B.; GUTTAG, J. V. **Abstraction and Specificarion in Program Development**. [S.l.]: MIT Press, 1986. ISBN 0-262-12112-3.

MACLEAN, A.; YOUNG, R. M.; MORAN, T. P. **DESIGN RATIONALE:** THE ARGUMENT BEHIND THE ARTIFACT. CH1'89 PROCEEDINGS. [S.l.]: [s.n.]. 1989.

MCGUINNESS, N. F. N. A. D. L. **Ontology Development 101: A Guide to Creating Your First Ontology**. [S.l.]. 2001.

MEDEIROS, A. P. D. **Kuaba: Uma Abordagem para Representação de Design Rationale para o Reuso de Designs baseados em Modelo**. Rio de Janeiro. 2006.

MEDEIROS, A. P. D.; SCHWABE, D. **Reuse, Ontology Driven Design Rationale**. Rio de Janeiro. 2006.

MIZOGUCHI, R.; IKEDA, M. **Towards Ontology Engineering**. Osaka: [s.n.], 1998.

MOORE, B. et al. **Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework**. [S.l.]: [s.n.], 2004.

MOORE, E. A.; AGOGINO, A. M. INFORM: An architecture for expert-directed knowledge acquisition. **International Journal of Man-Machine Studies**, 1987.

OMG. **MDA Guide Version 1.0.1**. [S.l.]: [s.n.], 2003.

OMG. **Meta Object Facility (MOF) Core Specification**. [S.l.]: [s.n.], 2006.

OMG. **Object Constraint Language**. [S.l.]: [s.n.], 2006.

OMG. **OMG Unified Modeling Language (OMG UML),Superstructure, V2.1.2**. [S.l.]. 2007.

POTTS, C.; BRUNS, G. **Recording the reasons for design decisions**. International Conference on Software Engineering. Los Alamitos, CA, USA: IEEE Computer Society Press. 1988. p. 418 - 427.

PRESUTTI, V.; GANGEMI, A. **Content Ontology Design Patterns as practical building blocks for web ontologies**. Barcelona, Spain: Springer-Verlag, 2008. ISBN ISBN:978-3-540-87876-6.

SÁNCHEZ, D.; MORENO, A. **Web-scale taxonomy learning**. Tarragona, Spain: [s.n.]. 2005.

SCHMID, H. **Part-Of-Speech Tagging with Neural Networks**. COLING 1994 Volume 1: The 15th International Conference on Computational Linguistics. Stuttgart, Germany: [s.n.]. 1994.

SCHNEIDER, L. **Designing Foundational Ontologies - The Object-Centered High-level Reference Ontology OCHRE as a Case Study**. Conceptual Modeling – 2003, 22nd International Conference on Conceptual Modeling. [S.l.]: Springer. 2003. p. 91-104.

SMITH, B. Fiat Objects, Topoi, p. 131-148., 2001.

STEINBERG, D. et al. **EMF:** Eclipse Modeling Framework. 2ª Edição. ed. [S.l.]: Addison-Wesley Professional, 2009.

USCHOLD, M.; GRUNINGER, M. **Ontologies:** Principles, Methods and Applications. [S.l.]: [s.n.], 1996.

USCHOLD, M.; KING, M. **Towards a Methodology for Building Ontologies**. IJCAI95 Workshop on Basic Ontological Issues in Knowledge Sharing. Montreal: [s.n.]. 1995.

VÖLKER, J. et al. **AEON - An Approach to the automatic evaluation of ontologies**. Applied Ontology. Amsterdam: IOS Press. 2008. p. 41-62.

W3C. RDF. **W3C**, 2004. Disponivel em: <http://www.w3.org/RDF/>. Acesso em: Agosto 2010.

W3C. Web Ontology Language (OWL). **W3C**, 2004. Disponivel em: <http://www.w3.org/2004/OWL/>. Acesso em: Agosto 2010.

ANEXO A – MODELO DE DADOS RELACIONAL

Este anexo apresenta as consultas SQLs executadas sobre a camada de persistência do modelo para geração do DPModel. Este modelo de dados é utilizado na implementação da proposta deste trabalho.

Entidade *Model*

Entidade que contém um identificador único para o modelo, o autor, a data de criação, a descrição e o nome.

```
SQL
SELECT mit.id, mif.author, mif.creationtime, mif.description, mif.name
        from modelinstance mit left outer join modelinformation mif
                on mit.modelinformation_id = mif.id
```

Entidade *Model Element*

Entidade que representa todos os elementos dos modelos, sejam relações ou tipos. Contém um identificador único, o identificador do modelo o qual a entidade pertence e o nome da entidade.

```
SQL:
SELECT     modelinstance_id model_id, instances_id id,              ore.value name
        FROM
                modelinstance_ontoeinstance m
                left outer join ontoeentity_ontoerestriction oo on (m.instances_id =
oo.ontoeentity_id)
                left outer join ontoerestriction ore on oo."restrictionsMap_uuid" = ore.uuid
        WHERE
                ore.name='name'
```

Entidade *Model Element Relationship*

Entidade que representa somente os relacionamentos dos modelos. Contém um identificador único, o modelo que contém a relação, e as entidades fonte e alvo da relação.

```
SQL:
SELECT        m.model_id,        m.id,        m.value,        source_id,        target_id
FROM        ontoerelationobject oe,        view.modelelement_vw m
where        m.id = oe.id
```

Entidade *Estereótipos*

Entidade que representa todas as metaclasses contidas em um metamodelo. O estereótipo utilizado na linguagem são essas metaclasses. Contém um identificador único, um nome amigável e um nome que é idêntico ao que aparece no metamodelo.

```
SQL:
select  id,     label,      name
from        ontoerelation
union
select       id,       label,       name
from        ontoeclass
```

Entidade *Classification*

Entidade que representa a classificação de um *model element*, ou seja qual o papel dentro de um padrão de projeto que a entidade está executando. Esta informação é útil para várias pesquisas como demonstrado durante o trabalho. Esta SQL é composta vários trechos que são unidos por uniões – em SQL o comando "UNION" – cada trecho extrai do modelo elementos que estão executando um determinado papel.

```
SQL:
select  modelelement_stereotype_vw.element_id, 11 role_id
from view.modelelement_vw,
      view.stereotype_vw, view.modelelement_stereotype_vw
where stereotype_vw.name = 'Phase' and
      modelelement_stereotype_vw.stereotype_id = stereotype_vw.id
      --PhaseSuperType
union
select distinct m.id, 10 role_id
from view.modelelement_vw m, view.stereotype_vw, view.modelelement_stereotype_vw
where
      m.id in (
            select m.source_id from view.modelelement_relations_vw  m,
                  view.stereotype_vw, view.modelelement_stereotype_vw
            where
                  stereotype_vw.name = 'Generalization' and
                  modelelement_stereotype_vw.stereotype_id = stereotype_vw.id and
                  m.id = modelelement_stereotype_vw.element_id
                  and m.target_id in (
                        select modelelement_stereotype_vw.element_id from
                              view.modelelement_vw, view.stereotype_vw,
view.modelelement_stereotype_vw
                        where
                              stereotype_vw.name = 'Role' and
                              modelelement_stereotype_vw.stereotype_id =
stereotype_vw.id
                  )
            )
union
```

```
	-- Role
select modelelement_stereotype_vw.element_id, 31 role_id
from view.modelelement_vw, view.stereotype_vw,
	view.modelelement_stereotype_vw
where stereotype_vw.name = 'Role' and
	modelelement_stereotype_vw.stereotype_id = stereotype_vw.id
	--RoleSuperType
union
select distinct m.id, 30 role_id
from view.modelelement_vw m, view.stereotype_vw,
	view.modelelement_stereotype_vw
where
	m.id in (
		select m.source_id
		from
			view.modelelement_relations_vw  m,
			view.stereotype_vw,
			view.modelelement_stereotype_vw
		where
			stereotype_vw.name = 'Generalization' and
			modelelement_stereotype_vw.stereotype_id = stereotype_vw.id and
			m.id = modelelement_stereotype_vw.element_id
			and m.target_id in (
				select modelelement_stereotype_vw.element_id
				from
					view.modelelement_vw,
					view.stereotype_vw,
					view.modelelement_stereotype_vw
				where
					stereotype_vw.name = 'Role' and
					modelelement_stereotype_vw.stereotype_id =
stereotype_vw.id
		)
	)
	-- Subkind
union
select modelelement_stereotype_vw.element_id, 21 role_id
from view.modelelement_vw, view.stereotype_vw, view.modelelement_stereotype_vw
where
	stereotype_vw.name = 'SubKind' and
	modelelement_stereotype_vw.stereotype_id = stereotype_vw.id
	--SubKindSuperType
union
select distinct
	m.id,20 role_id
from
	view.modelelement_vw m,
	view.stereotype_vw,
	view.modelelement_stereotype_vw
where
	m.id in (
		select
			m.source_id
		from
			view.modelelement_relations_vw  m,
```

```
                view.stereotype_vw,
                view.modelelement_stereotype_vw
        where
                stereotype_vw.name = 'Generalization' and
                modelelement_stereotype_vw.stereotype_id = stereotype_vw.id and
                m.id = modelelement_stereotype_vw.element_id
                and m.target_id in (
                        select modelelement_stereotype_vw.element_id
                        from view.modelelement_vw, view.stereotype_vw,
                                view.modelelement_stereotype_vw
                        where
                                stereotype_vw.name = 'SubKind' and
                                modelelement_stereotype_vw.stereotype_id =
stereotype_vw.id
                )
    )
```

Associação entre os Classifications

Esta SQL é responsável por extrair do modelo a relação entre entidades que atuam em uma mesma instância de um padrão de projeto.

```
OCL:
--Ser Superphase.
select (m.source_id || m.target_id || 10) id,  m.source_id, m.target_id, 10
association_type_id from
        view.modelelement_relations_vw  m,
        view.stereotype_vw,
        view.modelelement_stereotype_vw
    where
        stereotype_vw.name = 'Generalization' and
        modelelement_stereotype_vw.stereotype_id = stereotype_vw.id and
        m.id = modelelement_stereotype_vw.element_id
        and m.target_id in (
                select c.element_id target from
                view.classification_vw c
                where role_id = 11
        )
union
--Ser SuperSubKind
select (m.source_id || m.target_id || 20) id,  m.source_id, m.target_id, 20
association_type_id from
        view.modelelement_relations_vw  m,
        view.stereotype_vw,
        view.modelelement_stereotype_vw
    where
        stereotype_vw.name = 'Generalization' and
        modelelement_stereotype_vw.stereotype_id = stereotype_vw.id and
        m.id = modelelement_stereotype_vw.element_id
        and m.target_id in (
                select c.element_id target from
                view.classification_vw c
                where role_id = 21
        )
union
```

```
--Ser SuperRole
select (m.source_id || m.target_id || 30) id, m.source_id, m.target_id,30
association_type_id from
		view.modelelement_relations_vw  m,
		view.stereotype_vw,
		view.modelelement_stereotype_vw
	where
		stereotype_vw.name = 'Generalization' and
		modelelement_stereotype_vw.stereotype_id = stereotype_vw.id and
		m.id = modelelement_stereotype_vw.element_id
		and m.target_id in (
			select c.element_id target from
			view.classification_vw c
			where role_id = 31
		)
union
-- Phase Partition
select (m1.source_id || m2.target_id || 11) id, m1.target_id, m2.target_id , 11
association_type_id
from
(select m.source_id, m.target_id, m.name  from
		view.modelelement_relations_vw  m,
		view.stereotype_vw,
		view.modelelement_stereotype_vw
	where
		stereotype_vw.name = 'Generalization' and
		modelelement_stereotype_vw.stereotype_id = stereotype_vw.id and
		m.id = modelelement_stereotype_vw.element_id
		and m.target_id in (
			select c.element_id target from
			view.classification_vw c
			where role_id = 11
		)
) m1,
(select m.source_id, m.target_id, m.name from
		view.modelelement_relations_vw  m,
		view.stereotype_vw,
		view.modelelement_stereotype_vw
	where
		stereotype_vw.name = 'Generalization' and
		modelelement_stereotype_vw.stereotype_id = stereotype_vw.id and
		m.id = modelelement_stereotype_vw.element_id
		and m.target_id in (
			select c.element_id target from
			view.classification_vw c
			where role_id = 11
		)
) m2
where
m1.source_id=m2.source_id
and m1.target_id<> m2.target_id
and COALESCE(m1.name,'STANDARDGSET')= COALESCE (m2.name, 'STANDARDGSET')
union
-- SubKindPartition
select (m1.source_id || m2.target_id || 21) id, m1.target_id, m2.target_id , 21
```

```
association_type_id
from
(select m.source_id, m.target_id from
            view.modelelement_relations_vw  m,
            view.stereotype_vw,
            view.modelelement_stereotype_vw
      where
            stereotype_vw.name = 'Generalization' and
            modelelement_stereotype_vw.stereotype_id = stereotype_vw.id and
            m.id = modelelement_stereotype_vw.element_id
            and m.target_id in (
                  select c.element_id target from
                  view.classification_vw c
                  where role_id = 21
            )
) m1,
(select m.source_id, m.target_id from
            view.modelelement_relations_vw  m,
            view.stereotype_vw,
            view.modelelement_stereotype_vw
      where
            stereotype_vw.name = 'Generalization' and
            modelelement_stereotype_vw.stereotype_id = stereotype_vw.id and
            m.id = modelelement_stereotype_vw.element_id
            and m.target_id in (
                  select c.element_id target from
                  view.classification_vw c
                  where role_id = 21
            )
) m2
where
m1.source_id=m2.source_id
and m1.target_id<> m2.target_id
and COALESCE(m1.name,'STANDARDGSET')= COALESCE (m2.name, 'STANDARDGSET')
```

ANEXO B – REGRAS DE PESQUISAS PARA PADRÕES LÉXICOS

Neste anexo são apresentadas as entradas para pesquisas no subsistema de pesquisa por padrões léxicos. Este subsistema é alimentado por um arquivo XML que contém como principais dados:

- A variável que será substituída, entre a marcação *variable*
- As frases para pesquisa estão dentro da marcação *queries* e cada frase dentro da marcação *query*
- A expressão *regex* que será utilizada para filtrar após as frases passarem pelo POS Tagger, está dentro da marcação *expression*.

O arquivo é iniciado pela marcação <*patterns*>. A marcação <*pattern*> especifica os padrões para uma metapropriedade especificada pela propriedade *meta-property*. A marcação *corpus* específica um corpo de pesquisa que evidencia um valor para uma metapropriedade determinado pela propriedade *evidence*. A propriedade *linguística-filter* determina se o corpo de pesquisa passará por um POS Tagger e um filtro *regex.*

A expressões regulares ficam dentro da marcação <*regularexpression*>, pode haver mais de uma expressão regular dentro dessa marcação, neste caso a frase só precisará ser aceita por um dos filtros. Cada expressão está determinada pela marcação <*expression*>.

As pesquisas feitas no Google são especificadas pela marcação <*queries*>, cada pesquisa específica esta delimitada dentro de uma marcação <*query*>. As frases retornadas são a soma de todas as pesquisas.

Arquivo XML

```
<?xml version="1.0" encoding="UTF-8"?>
<patterns>
      <pattern meta-property="rigidity">
            <variable>-x-</variable>
            <corpus evidence="false" linguistic-filter="true">
                  <regularexpressions>
```

```
      <expression>[^?(There)]\bis\b.*\tno.*\tlonger.*?(NN|NP|NNS|NPS)\
t\b-x-\b\s+[^?(NN|NS|NPS|NNS)]</expression>
                  </regularexpressions>
                  <queries>
                        <query>is no longer a -x-</query>
                        <query >is no longer an -x-</query>
                  </queries>
            </corpus>

            <corpus evidence="false" linguistic-filter="true">
                  <regularexpressions>

      <expression>\bbecame\b.*?(a|an).*?(NN|NP|NNS|NPS)\t\b-x-
\b\s[^?(NN|NS|NPS|NNS)]</expression>
                  </regularexpressions>
                  <queries>
                        <query>became a -x-</query>
                        <query>became an -x-</query>
                  </queries>
            </corpus>
      </pattern>
      <pattern meta-property="dependence">
            <variable>-x-</variable>
            <corpus evidence="true" linguistic-filter="true">
                  <regularexpressions>

      <expression>\bcannot\b.*\bbe\b.*?(NN|NP|NNS|NPS)\t\b-x-
\b\s\w+\t\bwithout\b</expression>
                  </regularexpressions>
                  <queries>
                        <query>cannot be a -x- without</query>
                        <query>cannot be an -x- without</query>
                        <!-- <query>cannot be -x- without</query> -->
                  </queries>
            </corpus>
            <corpus evidence="true" active="true"  linguistic-
filter="true">
                  <regularexpressions>

      <expression>\bcannot\b.*\bbe\b.*?(NN|NP|NNS|NPS)\t\b-x-
\b\s\w+\t\bunless\b</expression>
                  </regularexpressions>
                  <queries>
                        <query>cannot be a -x- unless</query>
                        <query>cannot be an -x- unless</query>
                        <!-- <query>cannot be -x- without</query> -->
                  </queries>
            </corpus>
      </pattern>

</patterns>
```

ANEXO C – MODELOS UTILIZADOS PARA GERAR A BASE HISTÓRICA

Para que a aplicação gerasse as sugestões apresentadas no caso de uso demonstrado neste trabalho foi necessário popular o repositório do DPModel com alguns modelos. Estes modelos foram retirados de publicações e repositórios de ontologias e criados na ferramenta *OntoUML WebEditor*.

Neste anexo são demonstrados alguns modelos que foram inseridos na ferramenta antes da criação do caso de uso.

Domínio de Genealogia

Publicado em (GUIZZARDI, PIRES e SINDEREN, 2005)

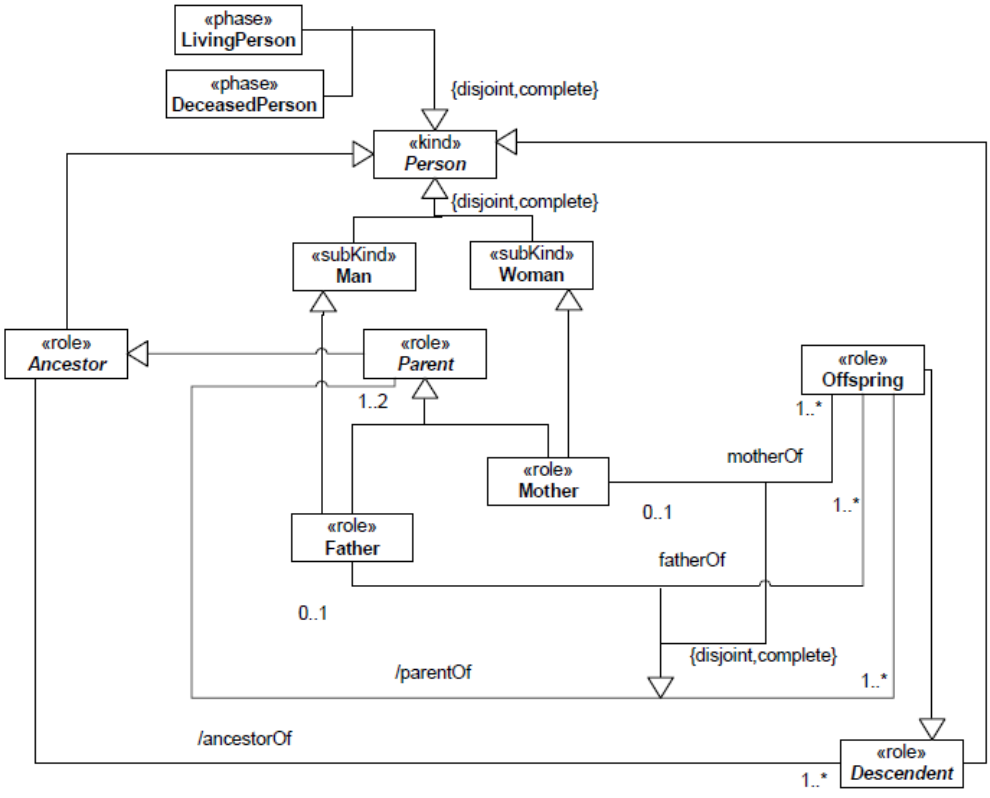

Figura 53 - Ontologia de domínio, acerca de Genealogia (GUIZZARDI, PIRES e SINDEREN, 2005)

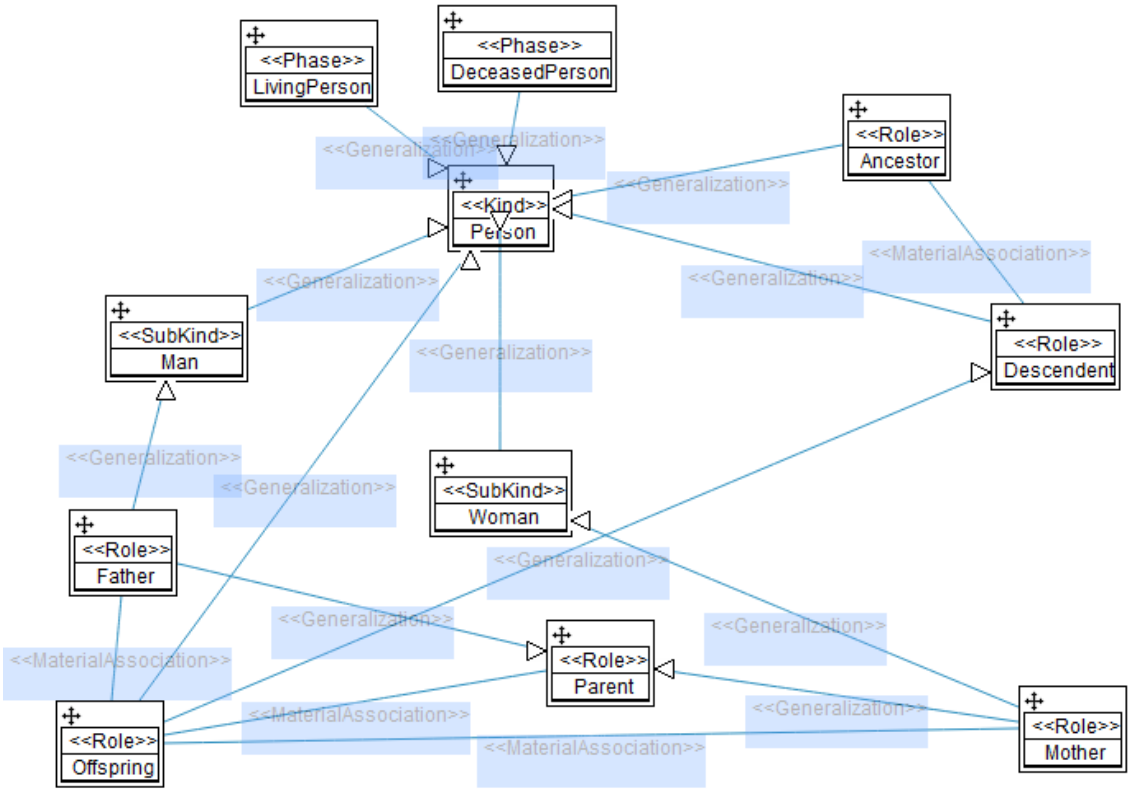

Figura 54 - Representação do modelo na Ferramentoa OntoUML WebEditor

Relações entre clientes corporativos e físico e uma organização

Modelo que demonstra as relações entre clientes corporativos e físicos e uma organização. Retirado de (GUIZZARDI, 2005).

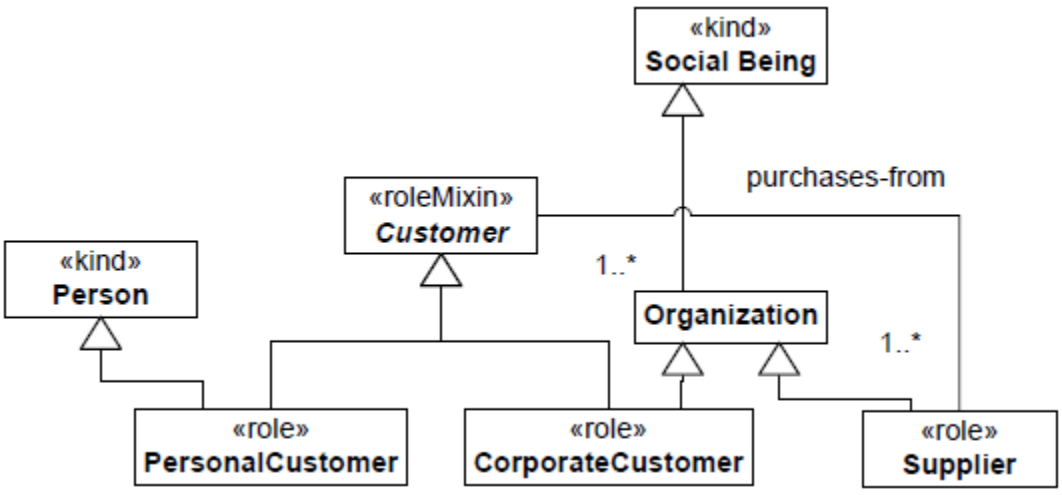

Figura 55 - Modelo retirado de (GUIZZARDI, 2005)

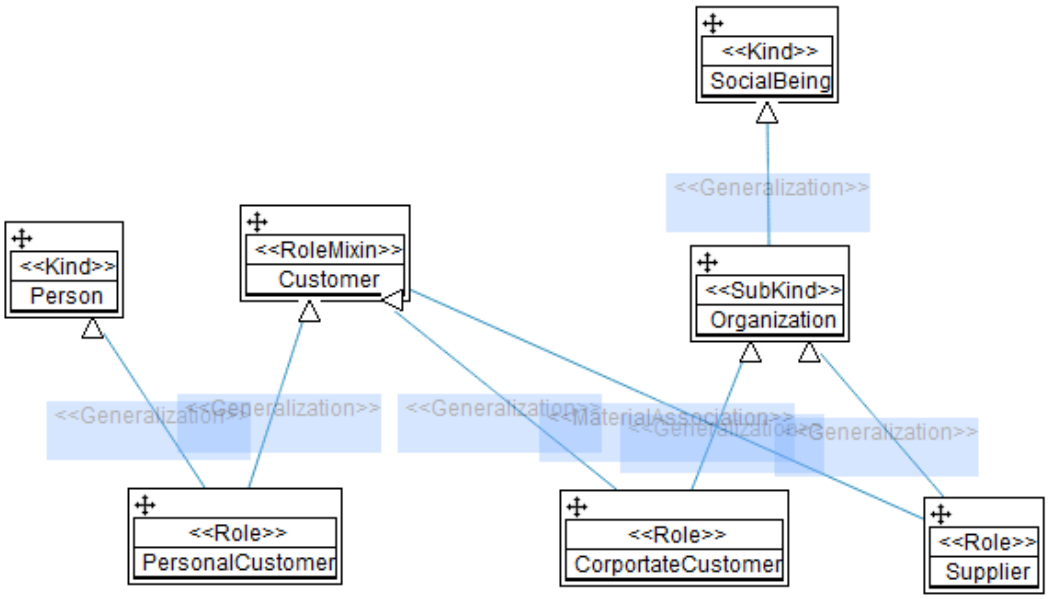

Figura 56 - Representação do modelo na Ferramentoa OntoUML WebEditor

Modelo Conceitual de uma Revista

Este modelo conceitual é parte de um modelo presente repositório de ontologias WebProtege (http://webprotege.stanford.edu/)

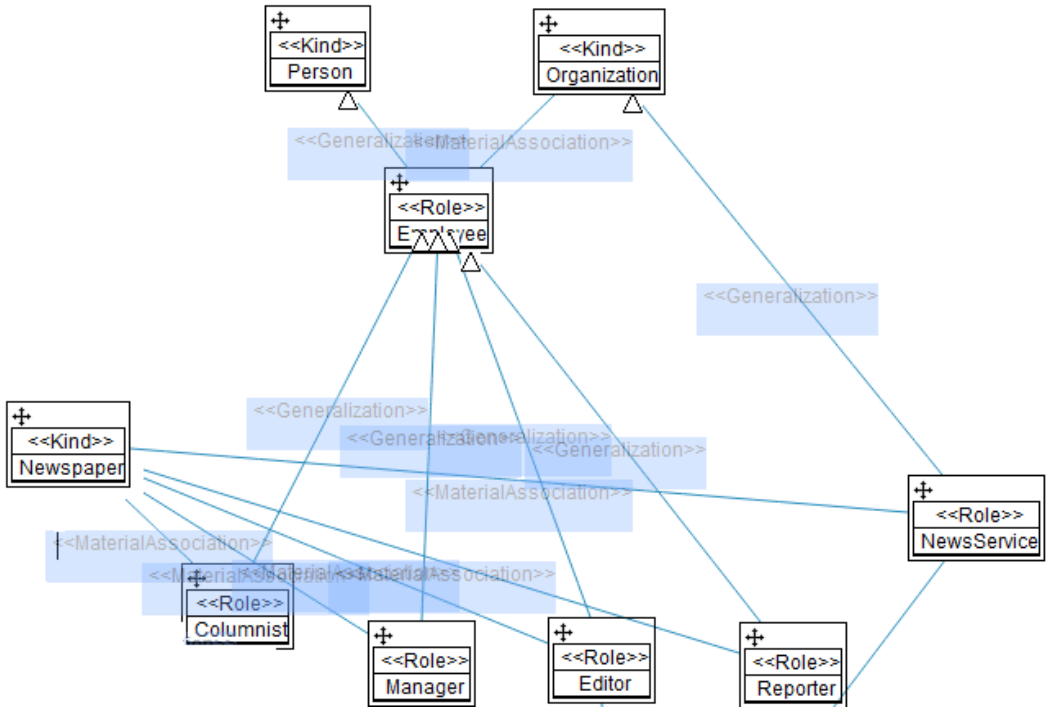

Figura 57 - Representação do modelo na Ferramenta OntoUML WebEditor

ANEXO D - CRIAÇÃO DE MODELOS NA FERRAMENTA

A ferramenta OntoUML WebEditor apresenta os modelos OntoUML como a maioria das ferramentas de modelagem UML. A barra lateral possui os construtores da OntoUML.

A paleta de ferramentas *OntoUML Concepts* possui todos os construtores que são classificadores, ao clicar em um desses construtores e clicar novamente na área de modelagem esse construtor é representado no modelo.

A paleta *OntoUML Relations* possui todos os construtores da linguagem que são relações, após clicar em um dos construtores é necessário clicar no classificador origem e arrastar até o classificador destino, para representar a relação.

Como explicado a validação do modelo é feita de duas maneiras, *live mode,* que é feita automaticamente sempre que uma ação sobre o modelo é executada e *batch mode,* que é a validação feita a pedido do usuário. A ferramenta também conta com um console OCL. Este console capacita fazer pesquisas OCL sobre o modelo instanciado.

Figura 58 - Barra de Ferramentas

A barra de ferramentas,Figura 58, contém uma lista de seleção – componente 1 – onde é possível escolher o desenho das relações. É possível três estados: *Straight*: uma relação é representada por uma linha reta; *Rectangular*: uma relação é representada por uma linha com pontos de quebra em 90º; *Curve*: as relações são representadas por linhas curvas, calculadas pela expressão de Bezier.

O botão com o ícone da lixeira – botão 2 da Figura 58 – apaga todo o modelo representado; O botão número 3 persiste o modelo. O botão número 4 executa as validações em modo Batch; O botão número 5 abre a possibilidade de fazer um upload de um modelo em ECore; O botão número 6 desfaz a última ação.

A barra de Menus, visível na Figura 59, contém duas opções File, Tools. A Barra de Menu File apresenta quatro opções:

- *New Model*: Saí do modelo atual e cria um novo modelo;

- *Load Model*: Saí do modelo atual e carrega um novo modelo;
- *Edit Model Information*: Abre o dialólgo *Model Description* onde é possível atribuir um nomo ao modelo e descreve-lo.
- *Export XMI File*: Salva o modelo como XMI, no formato Ecore.

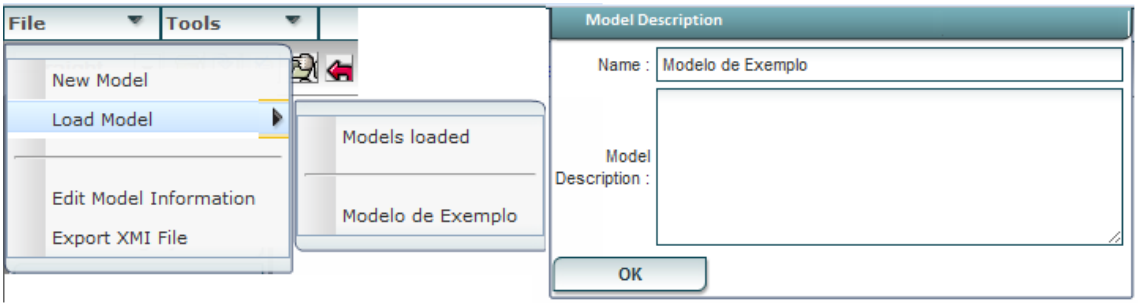

Figura 59 - Barra de Menus File e Dialogo Model Description

O Menu *Tools* contém a opção *Open OCL Console*, que abre um diálogo que possibilita executar pesquisas OCL sobre o modelo representado. E outras opções sobre o funcionamento do aplicativo.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo