Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Um Modelo Adaptativo de Objetos para Modelos Conceituais Baseados em UFO

Jideão José Vieira Filho

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Orientadora
Profa. Dra. Edna Dias Canedo

Brasília
2019

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

VV658m

Vieira Filho, Jideão José
Um Modelo Adaptativo de Objetos para ModelosConceituais Baseados em UFO / Jideão José Vieira Filho; orientador Edna Dias Canedo. -- Brasília, 2019.
139 p.

Dissertação (Mestrado - Mestrado em Informática) -- Universidade de Brasília, 2019.

1. Modelo Adaptativo de Objetos. 2. Ontologia de Fundamentação. 3. Unified Foundational Ontology. 4. Banco de Dados. I. Dias Canedo, Edna, orient. II. Título.

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Um Modelo Adaptativo de Objetos para Modelos Conceituais Baseados em UFO

Jideão José Vieira Filho

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Profa. Dra. Edna Dias Canedo (Orientadora)
CIC/UnB

Prof. Dr. João Paulo Andrade Almeida
Universidade Federal do Espírito Santo

Prof. Dr. Giancarlo Guizzardi
Universidade Federal do Espírito Santo

Profa. Dra. Genaina Nunes Rodrigues
Coordenador do Programa de Pós-graduação em Informática

Brasília, 10 de Dezembro de 2019

Dedicatória

Para Neuraci e Luana, principais amparos nesta caminhada.
Para Lauro, principal incentivador e mentor deste trabalho.

Agradecimentos

Primeiramente, agradeço à Deus pela saúde, pela paz e por me cercar de tantas pessoas queridas.

Agradeço à amada Luana Di Araújo pelo incondicional apoio nesta fase. Você foi extremamente importante neste período de tanta luta. Obrigado pelo apoio, pelo carinho, pela parceria.

Agradeço aos meus pais, Neuraci e Jideão, e ao meu irmão, Jonathas, pelo zelo, pelo cuidado, pela paciência e pelo amor. Especialmente, agradeço à minha mãe pela doação e pela confiança. Tenho muito orgulho da senhora e dificilmente chegaria aqui sem o seu apoio.

Agradeço imensamente ao amigo Dr. Lauro César Araujo pela sua entrega a este projeto. Obrigado pelo tempo disponibilizado e por me receber tantas vezes em seu lar para me orientar, aconselhar, apoiar. Sua entrega a este projeto, mesmo sem receber nada em troca, foi imprescindível para conseguir concluí-lo. Mais uma vez, muito obrigado.

Agradeço também ao amigo Dr. João Alberto de Oliveira pela cooperação técnica, pelos conselhos, pelo sentido prático que tanto ajudou neste trabalho.

Agradeço a minha orientadora, Dra. Edna Dias Canedo, pela oportunidade que me deu, pelo apoio e pela confiança durante a execução desta pesquisa.

Desejo também agradecer nominalmente ao amigo João Rafael Nicola pela paciência em me responder as tantas e tantas perguntas que lhe fazia. Sua contribuição para este trabalho foi fundamental para desatar nós que pareciam impossíveis.

Aos amigos Aline Leite e Marcos Leite, gostaria de agradecer por me acolherem no seu lar em um momento tão crítico da pesquisa.

À querida Bel Maciel e ao amigo Lázaro Araujo, agradeço pela compreensão e pelo apoio nesta jornada.

Ao amigo Ricardo da Silva Lima, muito obrigado pelas considerações e pelo convivo diário.

À querida Rosa de Araújo Santos, agradeço pela amizade e colaboração.

Também agradeço aos professores Dr. Giancarlo Guizzardi, Dra. Célia Ghedini Ralha e Dr. João Paulo Almeida por participarem das bancas de avaliação final e qualificação, pela disponibilidade, pelas críticas e sugestões de melhorias para este trabalho.

Aos amigos Paulo Estevão da Cruz Lima Junior e Leonardo Milhardes Mendes pelo interesse neste trabalho e pelas valorosas sugestões.

Ao prof. Dr. Carlos Galvão Pinheiro Junior, muito obrigado pelo encorajamento e pelos conselhos no início deste projeto.

Ao Instituto Legislativo Brasileiro, ao Prodasen e ao Senado Federal, agradeço pelo incentivo à pesquisa acadêmica como forma de aperfeiçoamento do cumprimento da missão institucional do Senado Federal.

Agradeço, por fim, a todas as pessoas envolvidas diretamente ou indiretamente neste projeto. Foram dois anos de muita luta, em que estive ausente em diversas oportunidades, mas em que sempre pude contar com o apoio de vocês.

Resumo

Alguns domínios complexos exigem alterações constantes de modelagem por possuírem muitas entidades, propriedades e relacionamentos pouco conhecidos nas fases iniciais de modelagem de software. Alterações constantes exigem criação de novas tabelas em banco de dados, alteração de código fonte, publicação das alterações etc. O custo de alteração de modelagem com um software em execução é mais caro quando comparado a alterações no início do ciclo de vida. Alterações constantes também tendem a inserir erros de modelagem, descrições inexatas e problemas de comunicação entre diferentes softwares, caso sejam feitas sem fundamentação ontológica consistente e clara. Por isso, este trabalho apresenta um modelo adaptativo de objetos (*Adaptive Object Model* (AOM)) para modelos conceituais baseados na *Unified Foundational Ontology* (UFO). Com esta abordagem, espera-se que alterações em tempo de execução não só sejam possíveis como também sejam feitas rapidamente e com baixo impacto. Além disso, por utilizar a UFO – uma ontologia de fundamentação sólida, que já se mostrou adequada para construção de ontologias de domínio robustas –, reduz-se riscos de alterações ontologicamente incoerentes, de problemas de interoperabilidade e de definições confusas.

Palavras-chave: Modelo Adaptativo de Objetos, Ontologia de Fundamentação, metamodelagem, modelagem conceitual, *Unified Foundational Ontology (UFO)*

Abstract

Some modeled domains are complex and require constant modeling changes because they are constituted of many entities, properties, and relationships but little is known about their details in the early stages of software development. Constant changes require creating new database tables, changing source code, deploying new software, and so on. The cost of model changes during software execution is more expensive than in early life cycle phases. Constant changes also tend to insert modeling errors, inaccurate descriptions and communication problems if they are made without consistent and clear ontological foundations. Therefore, this work presents an Adaptive Object Model (AOM) for conceptual models based in the Unified Foundational Ontology (UFO). With this approach, it is expected that runtime changes should be possible without code and databases changes, but also should be done quickly and with low impact. In addition, using UFO – a sound foundational ontology that is suitable for building robust domain ontologies –, it should reduce the risk of ontologically incoherent changes, interoperability problems and confusing definitions.

Keywords: *Adaptive Object Model, Foundational Ontology, metamodeling, conceptual modeling, Unified Foundational Ontology (UFO)*

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AOM *Adaptive Object Model.*

BFO *Basic Formal Ontology.*

BLOB *Binary Large Object.*

BORO *Business Objects Reference Ontology.*

BWW *Bunge, Wand and Weber.*

CLOB *Character Large Object.*

CNPq Conselho Nacional de Desenvolvimento Científico e Tecnológico.

DESO *Digital Evidence Semantic Ontology.*

DOLCE *Descriptive Ontology for Linguistic and Cognitive Engineering.*

EER *Enhanced Entity–Relationship.*

GFO *Generalized Formalized Ontology.*

GOL *General ontological language.*

ISM *Implementation Specific Model.*

MDD *Model Driven Development.*

Milan *Associazione Calcio Milan.*

MLT *Multi-Level Theory.*

NEMO Núcleo de Estudos em Modelagem Conceitual e Ontologias.

NoSQL *Not Only SQL.*

ORM *Object-Relational Mapping.*

OWL *W3C Web Ontology Language.*

PIM *Platform Independent Model.*

PSG *Paris Saint-Germain.*

PSM *Platform Specific Model.*

RDB *Relational Database.*

RUP *Rational Unified Process.*

SOC *Separation of Concerns.*

SQL *Structured Query Language.*

SUMO *Suggested Upper Merged Ontology.*

UFES Universidade Federal do Espírito Santo.

UFO *Unified Foundational Ontology.*

UML *Unified Modeling Language.*

UnB Universidade de Brasília.

VAR *Video Assistant Referees.*

varchar *Variable Character Field.*

Lista de Códigos

Capítulo 1

Introdução

Produzir software envolve compreender o domínio para o qual a aplicação é construída. Porém, o software produzido representa apenas parte do domínio analisado, pois não é factível especificar todas as entidades de um domínio e seus atributos em detalhes, exceto em contextos muito simples e em situações hipotéticas ou artificiais. Por isso, domínios reais normalmente são complexos e abordagens tradicionais de desenvolvimento de sistemas tendem a gerar modelos lógicos e de banco de dados com centenas de entidades e tabelas. Por exemplo, este é o caso do domínio de gestão das normas jurídicas, visto que um arcabouço normativo regula inumeráveis aspectos da vida social e física.

É desafiador construir software para situações em que a compreensão do domínio é gradual, à medida que o sistema é construído. A longo prazo, muitas alterações de projeto são necessárias para refletir os detalhes esclarecidos sobre o domínio, o que se repercute em frequentes alterações de código-fonte, de criação e alteração de tabelas de banco de dados etc. Para Atkinson e Kuhne [1, p. 37], o nível de produtividade de software no longo prazo depende da longevidade do artefato de software. Quanto mais tempo o artefato permanece útil, maior o retorno derivado do esforço dispensado na criação do mesmo. Assim, se o artefato de software é estável, propenso a poucas modificações, o prognóstico de produtividade de software a longo prazo tende a ser melhor.

O desejo de desenvolver um arcabouço estável para a construção de software em domínios amplos e complexos[1] encoraja a realização desta pesquisa.

[1]No contexto desta pesquisa, domínios amplos e complexos são aqueles que possuem muitos detalhes, são propensos a sofrer alterações frequentes, cujas entidades pertencem a vários subdomínios e são pouco conhecidas no momento da modelagem conceitual.

1.1 Problema de Pesquisa

A modelagem do domínio da gestão de atos normativos é uma atividade descritiva com muitos detalhes e propensa a alterações frequentes. Representar com precisão todos os detalhes, propriedades e relacionamentos das entidades do domínio é impraticável. Por isso, espera-se a ocorrência de alterações contínuas de modelo nas fases finais de desenvolvimento de software e ao longo do ciclo de manutenção.

Em 1973, o Senado Federal brasileiro começou a tratar a informação legislativa e jurídica com recursos tecnológicos e computacionais [2]. A estimativa daquela época era de que o número de textos legais girava em torno de 110.000 Normas[2] [3]. Segundo do Amaral et al. [4], o ritmo de elaboração de Atos Normativos cresceu a partir de então: desde a promulgação da Constituição Federal de 1988, em 5 de outubro daquele ano, até 30 de setembro de 2018, foram editados 5.876.143 de textos legais no Brasil (nas três esferas: Federal, Estadual e Municipal), ou seja, uma média de 536 por dia (774, se considerarmos apenas dias úteis). Cada norma possui em média 3.000 palavras e apenas 4,13% dessas normas não sofreram alteração.

De acordo com do Amaral et al. [4], apenas no âmbito federal, editaram-se 166.241 normas desde a promulgação da Constituição Federal, passando por uma constituição, seis emendas constitucionais de revisão, cem emendas constitucionais, duas leis delegadas, 104 leis complementares, 5.967 leis ordinárias, 1.461 medidas provisórias originárias, 5.491 reedições de medidas provisórias, 12.643 decretos federais e 140.466 normas complementares (portarias, instruções normativas, ordens de serviço, atos declaratórios, pareceres normativos etc.) (conforme apresentado na Tabela 1.1).

Os textos legais possuem relacionamentos com uma quantidade inumerável de entidades. Por exemplo, atos jurídicos estabelecem instituições jurídicas, definem categorias, regulam seres, materializam entidades. A quantidade de seres, categorias e instituições jurídicas referenciadas em normas é relativamente grande, visto a amplitude do alcance dos atos legais e regulatórios.

Relações intra-textos legais também são frequentes e em grande quantidade: atos jurídicos são constantemente alterados, revogados, revigorados, vetados total ou parcialmente por outros atos jurídicos. Além disso, textos legais possuem um ciclo de vida: sua vigência e eficácia são dependentes de eventos como a publicação do ato, ou seja, existem também relacionamentos dos textos legais com eventos do mundo real. Ademais, com tantas normas no ordenamento jurídico brasileiro, é difícil determinar quais estão em vigor [2].

Apesar das dificuldades, é natural e desejável utilizar a tecnologia da informação para dar organicidade ao domínio jurídico. Entretanto, como existe uma distância entre a

[2]Provavelmente esta estimativa refere-se apenas às Normas Federais [2].

Tabela 1.1: Quantidade de Normas Editadas desde a promulgação da Constituição Federal de 1988 (Fonte: [4])

NORMAS FEDERAIS	**GERAIS**	**TRIBUTÁRIAS**
Constituição Federal	1	1
Emendas Constitucionais de Revisão	6	-
Emendas Constitucionais	100	16
Leis Delegadas	2	-
Leis Complementares	104	46
Leis Ordinárias	5.967	1.147
Medidas Provisórias Originárias	1.461	246
Medidas Provisórias Reeditadas	5.491	1.674
Decretos Federais	12.643	1.669
Normas Complementares	140.466	27.138
Total	166.241	31.937
Média por Dia	15,17	2,92
Média por Dia Útil	21,90	4,21

aplicação e os requisitos do domínio, normalmente o processo de projeto de software envolve várias etapas, em que cada etapa trata de *abstrações*[3] de um subconjunto limitado de questões de projeto (*Separation of Concerns* (SOC)) [5, p. 9]. Nesse aspecto, modelos são abstrações comumente utilizadas para representar um domínio [6]. Os modelos de software são descritivos da realidade representada e prescritivos para o sistema construído (vide Seção 2.1).

O domínio de normas jurídicas é um exemplo que tende a sofrer muitas alterações, pois os modelos seriam atualizados constantemente. Em uma abordagem tradicional de construção de software, por exemplo, a identificação de novas entidades implicaria na necessidade de extensão do banco de dados e alteração de código-fonte, o que frequentemente é trabalhoso [7]. Além disso, modelos maiores tendem a implicar em problemas de interoperabilidade e em definições confusas [8].

Logo, entende-se a gestão de atos normativos e suas relações como uma atividade essencialmente descritiva em que detalhes de propriedades e relacionamentos podem ser retratados de formas potencialmente inéditas, provavelmente incompletas e propensas a muitas alterações. Alterações frequentes resultam em problemas de manutenção de sistemas, visto que mudanças de modelos de software são complexas, principalmente mudanças de mais baixo nível.

[3]Abstração é o processo (ou o resultado do processo homônimo) de analisar certa entidade apenas pelas características consideradas relevantes por determinado ponto de vista [5, p. 10].

Apesar da motivação desta pesquisa ser a representação do domínio de atos jurídicos, o problema e a solução não se limitam ao domínio citado: espera-se que outros domínios de endurantes modelados com a *UFO*-A possam se beneficiar da abordagem proposta neste trabalho.

1.2 Justificativa

Nesta Seção, apresenta-se porque é cientificamente relevante resolver o problema apresentado na Seção 1.1 e relatam-se os motivos pela escolha da abordagem utilizada neste trabalho.

1.2.1 Por que a pesquisa é relevante?

De acordo com o *Rational Unified Process* (RUP) [9], para a construção de software, pelo menos duas fases de modelagem são necessárias (considerando que a modelagem do negócio nem sempre é realizada): a primeira na disciplina de requisitos, em que é realizada uma modelagem conceitual específica do domínio do software; a segunda na disciplina de Análise e Design, na qual o próprio software é especificado.

Construir sistemas demanda tempo, recursos humanos, físicos e financeiros. O custo de alteração do software aumenta à medida que se avança no ciclo de vida da aplicação: alterações de modelagem nas fases mais avançadas do desenvolvimento de software (implementação, testes e manutenção) são mais caras que nas fases iniciais (modelagem conceitual, análise e projeto) [10–16].

As Figuras 1.1 e 1.2 apresentam duas perspectivas do custo de alterações/correções em um software ao longo do tempo. Na Figura 1.1, é possível perceber que a correção de um erro quando o software está em manutenção chega a ser cem vezes maior que a mesma alteração feita na fase de projeto [16]. A Figura 1.2 apresenta, sob a perspectiva de projetos (ou seja, não engloba a fase de manutenção e pode abranger projetos que não sejam de software), o rápido crescimento dos custos enquanto decrescem as incertezas e os riscos, visto que os objetivos do projeto ficam mais claros com o desenvolvimento do mesmo [17].

Conforme descrito na Seção 1.1, nos casos em que detalhes de entidades do domínio são numerosos e pouco conhecidos, se existir a possibilidade de evolução do software alterando-se apenas o seu modelo de objetos[4], sem necessidade de alteração de tabelas de banco de dados nem de código-fonte, o impacto das alterações seria menor e recursos poderiam ser alocados em outras tarefas.

[4]modelo que contém as categorias de domínio. Vide Subseção 2.1.2

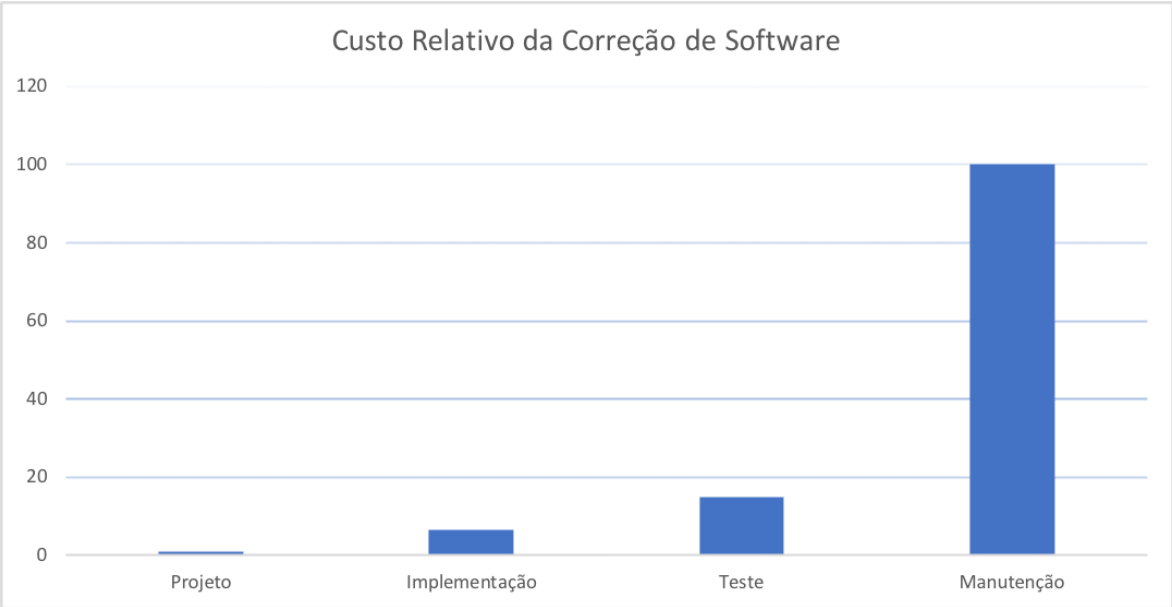


Figura 1.1: Custo relativo da correção de defeitos por fase de desenvolvimento de software (Fonte: adaptada de [16])

É possível evoluir software alterando-se apenas o modelo de objetos por meio de um tipo especial de meta-arquitetura[5] chamada de arquitetura *Adaptive Object Model* (AOM)[6] [18, 19].

Yoder e Johnson [18, p. 2, tradução nossa] apresentam a seguinte definição para AOMs:

> Um Modelo Adaptativo de Objetos é um sistema que representa classes, atributos, relacionamentos e comportamentos como metadados. Este sistema baseia-se em instâncias ao invés de classes. Usuários podem mudar os metadados (modelo de objetos) para refletir mudanças no domínio. Estas mudanças alteram o comportamento do sistema. Em outras palavras, o sistema armazena seu modelo de objetos em um banco de dados e o interpreta[7]. Consequentemente, o modelo de objetos é adaptável. Quando a informação descritiva é modificada, o sistema imediatamente reproduz essas mudanças[8].

Um AOM possui dois modelos de objetos: o *modelo de objetos da aplicação* e o *modelo de objetos de domínio*. O modelo de objetos de domínio contém as categorias de domínio,

[5] Arquitetura projetada para se adaptar a mudanças de requisitos de usuário em tempo de execução por meio da recuperação de informações descritivas que podem ser computacionalmente interpretadas em tempo de execução [18].

[6] Modelo Adaptativo de Objetos, em português

[7] No contexto desta dissertação, *interpretar* refere-se a interpretação computacional

[8] *An Adaptive Object-Model is a system that represents classes, attributes, relationships, and behavior as metadata. The system is a model based on instances rather than classes. Users change the metadata (object model) to reflect changes in the domain. These changes modify the system's behavior. In other word, the system stores its Object-Model in a database and interprets it. Consequently, the object model is adaptable; when the descriptive information is modified, the system immediately reflects those changes*

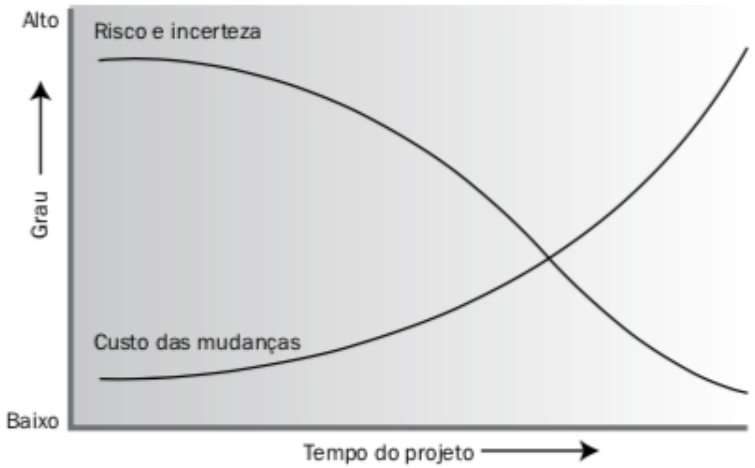


Figura 1.2: Impacto das variáveis risco, incerteza e custo de mudanças ao longo do tempo (Fonte: [17])

é adaptável e sofre alterações constantes. O modelo de aplicação é metamodelado, contém as metaclasses necessárias para interpretação computacional do modelo de objetos, deve ser estável e sujeito a poucas alterações. Deste ponto em diante, utiliza-se o termo *modelo de objetos* para referência ao *modelo de objetos de domínio*. Já os termos *modelo adaptativo* e *modelo de aplicação* referem-se ao *modelo de objetos da aplicação*. AOMs são descritos na Subseção 2.1.2.

Todo software apresenta conceitos de domínio e faz algum compromisso ontológico [20]. Um compromisso ontológico é "um relato semântico parcial da conceituação desejada de uma teoria lógica" [21, tradução nossa], ou ainda, "os objetos comuns à ontologia que cumpre alguma teoria controlada" [22, tradução nossa, p. 631]. Uma conceituação, por sua vez, é uma estrutura semântica que retrata um sistema conceitual particular e representa parte da realidade [21]. Ontologia, no sentido filosófico, é o estudo o ser. Logo, um ontologista se preocupa com perguntas como: o que existe ou quais classes de entidades existem [23]?

Como espera-se que o *modelo de objetos* seja alterado frequentemente, é importante fundamentá-lo em conceitos ontológicos sólidos, de modo a restringir problemas de modelagem ou problemas de alteração de modelo. Por isso, este trabalho utiliza-se da *Unified Foundational Ontology* (UFO) para o desenvolvimento de um modelo de objetos bem fundamentado ontologicamente. A UFO é uma ontologia de fundamentação robusta resultante de estudos em áreas como metafísica descritiva, psicologia cognitiva, linguística. Apresentam-se detalhes sobre a UFO nas Seções 2.5, 2.6 e 2.7.

1.2.2 Por que criar um AOM?

A construção do AOM visa obter um metamodelo de banco de dados estável para ser utilizado em diferentes domínios com modelos conceituais que seguem os axiomas da UFO.

Por um metamodelo estável, entende-se um metamodelo susceptível a poucas mudanças, ou seja, um metamodelo que possa ser reutilizado em diferentes domínios, idealmente sem alterações. Para permitir a reutilização em domínios diferentes, o metamodelo deve ser geral, extensível e com alto grau de abstração [5].

Um metamodelo geral é obtido por meio da generalização dos requisitos de usuário e objetivos de projeto ou por meio da antecipação de objetivos ou requisitos futuros. A extensibilidade é a característica de um (meta)modelo permitir extensões futuras [5].

Por alto grau de abstração entende-se que o processo de abstração será feito de modo a abranger o máximo possível de entidades, resultando assim, em um conjunto menor de metapropriedades compartilhadas entre entidades.

Desta forma, para se conseguir um metamodelo geral, extensível e que possua propriedades que sejam comuns a todas as classes de indivíduos, este trabalho opta pela utilização de um modelo adaptativo de objeto como modelo lógico. O modelo adaptativo de objeto é um metamodelo geral e bem fundamentado que permite modelos específicos de domínio serem implementados sobre o mesmo. Alterações no modelo específico de domínio não exigem modificações no modelo lógico.

1.2.3 Por que ontologias?

Não há como se esquivar de fazer algum compromisso ontológico na representação de abstrações para construção de modelos de software.

Abstrações são fundamentais para o processo de construção de software. O objetivo dos métodos utilizados em Engenharia de Software é abstrair os principais conceitos de um domínio, representá-los apropriadamente e transformá-los corretamente de modo a representar modelos do mundo real com modelos computacionais [6].

Na filosofia, existe um problema ontológico associado ao conceito de abstração que é tratado desde a Grécia antiga por Platão e Aristóteles e muito debatido na idade média: o problema dos universais. A filosofia tradicional diferencia universais (abstratos) dos particulares (concretos) [6]. Diferentes correntes da filosofia tratam universais de maneira distintas. Nominalistas, por exemplo, tratam universais como um termo. Afirmam que não existem na realidade. Já os realistas, afirmam que universais de fato existem. Esta discussão está fora do escopo deste trabalho[9].

[9]Mais detalhes podem ser encontrados em Klima [24] e Loux e Crisp [25].

Apesar de este trabalho não estar preocupado com questões filosóficas ou ontológicas relacionadas aos universais, há uma necessidade lógica de construir um arcabouço que permita a interpretação de modelos para resultar no sistema de informação. Por isso, este trabalho assume as mesmas posições adotadas pela UFO, já que esta ontologia de fundamentação será utilizada como base para a construção do modelo de objetos. Neste sentido, "universal é algo que pode ser instanciado por diferentes entidades"[10] [26, tradução nossa]. Um exemplo de universal seria *Universidade*. A *Universidade de Brasília (UnB)* é um particular de *Universidade*, ou seja, o particular *UnB* é uma instância do universal *Universidade*.

Essa distinção é muito importante para a Engenharia de Software, conforme constatam Pastor e Molina [6, p. 4, tradução nossa, grifo nosso][11].

> É importante que, no campo da Engenharia de Software, os modelos mentais a partir dos quais um Esquema Conceitual é construído representem precisamente os universais que participam do sistema analisado.
>
> Todo produto de software que resulta de um processo de transformação constituirá um mundo formado por particulares que serão instâncias dos universais correspondentes. Portanto, independentemente de você se considerar um realista, um nominalista ou um conceitualista, os mundos virtuais que são construídos a partir de um Esquema Conceitual serão sempre a representação dos universais considerados relevantes. Nesses mundos virtuais, nunca haverá um particular que não seja uma instância de seu universal correspondente.
>
> Implementações confiáveis podem ser geradas apenas pela especificação correta de universais. Em outras palavras, para se construir corretamente Sistemas de Software, é necessário especificar de forma precisa e não ambígua os universais e seus relacionamentos que são considerados relevantes. Isso resultará em um esquema conceitual útil. Para tanto, precisamos definir uma **ontologia** em que os conceitos relevantes para a especificação de universais estejam perfeitamente definidos.

Logo, toda modelagem de sistema faz algum compromisso ontológico [27]. Não importa como é feita a modelagem, ela sempre é retratada por uma ontologia. "Pois, se anti-

[10] *A universal is something that can be instantiated by different entities.*

[11] *... it is significant that, in the field of Software Engineering, the mental models from which a Conceptual Schema is built must represent precisely those universals that participate in the analysed system.*

Every software product that results from a transformation process will constitute a world formed by particulars that will be instances of the corresponding universals. Therefore, independently of whether you consider yourself to be a realist, a nominalist or a conceptualist, virtual worlds that are built from a Conceptual Schema will always be the representation of the universals considered to be relevant. In these virtual worlds, there will never be a particular that is not an instance of its corresponding universal [...].

Reliable implementations can be generated only by correctly specifying universals. In other words, to correctly build Software Systems, it is necessary to precisely and non-ambiguously specify the universals and their relationships that are considered to be relevant. This will result in a useful Conceptual Schema. In order to do so, we need to define an ontology *in which the relevant concepts for specifying universals are perfectly defined*

ontologia não consegue dissociar-se de ontologia, o que é senão uma ontologia ruim?[12]" [28, p. 59, tradução nossa]. Portanto, todo software conterá uma ontologia, independente de o engenheiro de software estar ciente ou não disso. Este trabalho trata este fato de maneira consciente, de modo a termos um modelo de objetos fundamentado em compromissos ontológicos sólidos.

1.2.4 Por que utilizar UFO?

O modelo de domínio construído sobre o metamodelo também deve ser estável, mas permitir constantes evoluções e extensões. Para isso, é importante ser alicerçado em conceitos sólidos. Conceitos sólidos dependem de uma ontologia de fundamentação embasada em princípios filosóficos seguros e consistentes.

Segundo Guizzardi et al. [29], "Ontologias de Fundamentação são sistemas de categorias filosoficamente bem fundamentados e independentes de domínio que têm sido utilizados com sucesso para melhorar a qualidade de linguagens de modelagem e modelos conceituais".

Araujo [30] defende que sistemas de informação mais especializados precisam de um sistema de classificação mais amplo que permita a expansão de conceitos por meio de processos como composição, derivação e especialização. E esse é o papel de Ontologias de Fundamentação: ser a base para ontologias de domínio mais especializadas. Em suma, ontologias de fundamentação são ontologias de alto nível que servem de referência para criação e evolução de ontologias *core* e de domínio. Muitas ontologias de fundamentação foram criadas, especialmente nas últimas décadas, sendo a UFO uma delas. A Tabela 1.2 apresenta outros exemplos de ontologias de fundamentação.

A UFO é uma ontologia de fundamentação considerada bem fundamentada e consolidada pela literatura [31]. Ela foi proposta como um esforço de síntese de uma seleção de ontologias de fundamentação. Baseia-se em um número de teorias das áreas de Ontologias Formais, Lógica Filosófica, Filosofia da Linguagem, Linguística e Psicologia Cognitiva [29]. Esta ontologia é aplicada com sucesso no projeto, avaliação e integração de modelos de linguagens de modelagem conceitual, assim como para fornecer semântica do mundo real para os elementos da modelagem [29, 32–35].

Um indício da qualidade das ontologias de fundamentação é sua utilização. Verdonck e Gailly [31] realizaram um estudo da frequência de referências em artigos científicos às principais ontologias de fundamentação conhecidas pela comunidade acadêmica. Os autores analisaram uma amostra de duzentos artigos e os resultados estão apresentados na Tabela 1.2.

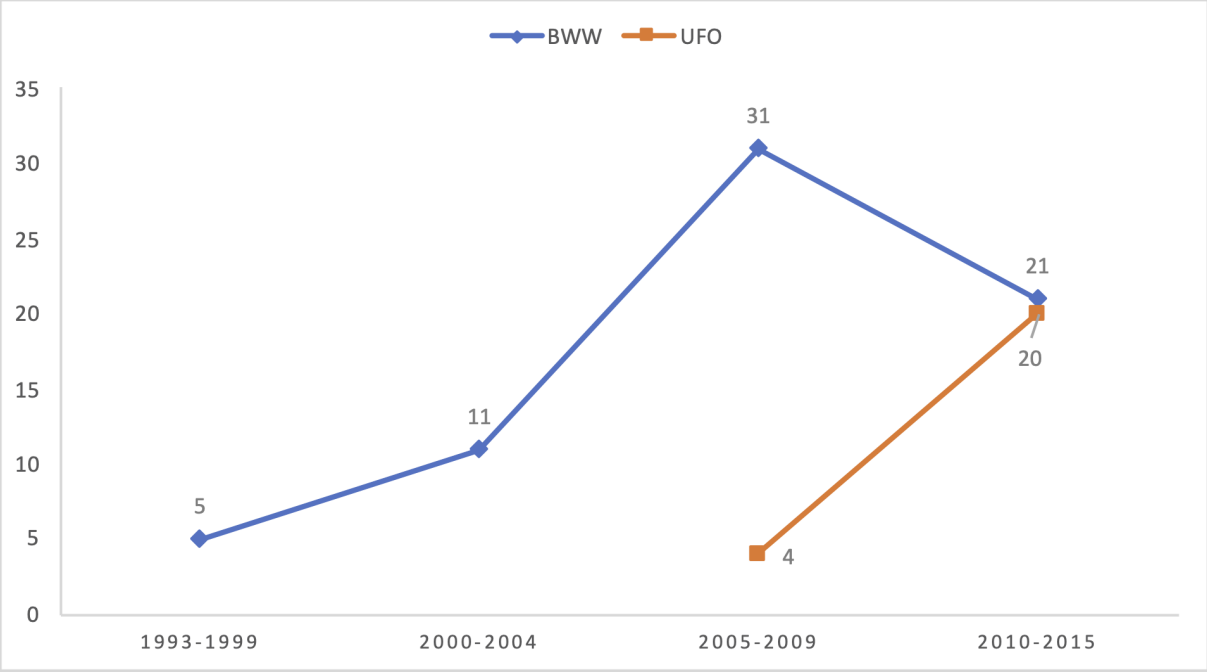
[12] *For if anti-ontology fails to dissociate itself from ontology, what is it but bad ontology?*

Tabela 1.2: Frequência de referências a ontologias de fundamentação na literatura. (Fonte: Adaptada de [31])

Ontologia de Fundamentação	**Referências**
Bunge, Wand and Weber (BWW) [36–38]	68
Unified Foundational Ontology (UFO)	24
Generalized Formalized Ontology (GFO) [39, 40]	4
Digital Evidence Semantic Ontology (DESO) [41, 42]	3
Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [43–45]	3
Ontologia de Chisholm	2
Suggested Upper Merged Ontology (SUMO) [46]	2
Business Objects Reference Ontology (BORO)	1
Basic Formal Ontology (BFO)	1
Ontologia de Searle	1

Observa-se que dentre as ontologias citadas, a UFO é a segunda mais referenciada, atrás apenas da ontologia *Bunge, Wand and Weber* (BWW) [36–38], criada por Yair Wand e Ron Weber e fundamentada no trabalho do filósofo argentino Mário Bunge [47]. Entretanto, ao se analisar a distribuição temporal das referências, percebe-se, conforme ilustrado na Figura 1.3, que a partir de 2010, cinco anos após o surgimento da UFO, iniciou-se um processo de progressiva preferência da UFO em detrimento à BWW.

Figura 1.3: Utilização da BWW e da UFO entre 1993 e 2015 (Fonte: Adaptada de [31])



A BWW foi provavelmente a primeira tentativa de desenvolver uma fundamentação ontológica para modelagem conceitual. Ela foi alicerçada na ontologia de Bunge [47], o qual, tinha como principal objetivo construir uma ontologia de ciências. Logo, a sua ontologia possui algumas limitações para modelagem conceitual. A ontologia de Bunge foi construída no domínio do mundo material. Ela consiste em objetos materiais que possuem propriedades físicas independentes da percepção humana. Ela não abrange objetos conceituais, os quais são necessários para representar conceitos fundamentais para a comunicação humana [48].

A modelagem conceitual, por sua vez, precisa tratar da representação de aspectos do mundo físico e social de modo a melhorar a comunicação e o entendimento de determinado tema [49]. Deste modo, para desenvolver fundamentações ontológicas para modelagem conceitual, deve-se observar aspectos de cognição e linguística humana profundamente.

Neste sentido, a UFO engloba teorias de estruturas de tipos e taxonômicas [50, 51] conectada a uma teoria de identificadores de objetos (incluindo uma semântica formal em lógica formal quantificada de sortais [52]), relações todo-parte [53, 54], propriedades intrinsecamente particularizadas, atributos e espaços de valores de atributos [55, 56], relações e propriedades relacionais particulares [57, 58] e papéis [59], dentre outras [30, cap. 4]. Por isso, a UFO mostrou-se uma ontologia mais adequada que a BWW para a modelagem conceitual proposta neste trabalho.

Outra motivação para a escolha da UFO foram suas aplicações práticas com sucesso e documentadas em vários domínios. Alguns exemplos são[13]:

- Gerenciamento de conteúdo midiático do portal Globo.com[14], das Organizações Globo [61];
- Gestão da biodiversidade [62];
- Modelagem de reservatórios de petróleo [63];
- Eletrofisiologia do coração [64].

Portanto, por ser uma das ontologias de fundamentação mais utilizadas, por possuir base filosófica e cognitiva sólida, por ter sido aplicada em domínios diversos e por tratar dos conceitos necessários para o desenvolvimento deste trabalho, a UFO foi escolhida como base para construção da ontologia do domínio de gestão de normas jurídicas proposta.

[13] Uma lista mais ampla de exemplos pode ser encontrada em [60].
[14] `http://globo.com`.

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo geral deste trabalho é apresentar um Modelo Adaptativo de Objetos (*Adaptive Object Model* (AOM)) para interpretação computacional de modelos de objetos construídos com a UFO.

1.3.2 Objetivos Específicos

Para atingir o objetivo geral deste trabalho, o modelo de aplicação deve oferecer:

1. expressividade suficiente para suportar o modelo de objetos baseado em um fragmento da UFO-A e da UFO-B;
2. conformidade com as restrições de axiomas relacionados a um fragmento da UFO-A e da UFO-B utilizado. De UFO-B, apenas eventos atômicos fazem parte do escopo deste trabalho;
3. capacidade de representar mudanças no tempo e diferenciar aspectos ontológicos (eventos de mundo) de aspectos epistemológicos (eventos de sistema)[15].

1.4 Resultados Esperados

Ao se criar um modelo adaptativo de objetos para interpretar modelos conceituais fundamentados na UFO, as principais contribuições esperadas são:

- permitir evoluções no modelo de objetos e aplicá-las ao software em produção sem alteração de código-fonte ou de tabelas do banco de dados;
- possibilitar alteração do modelo de objetos sem necessidade de reiniciar a aplicação;
- construir modelos de objetos sólidos;
- restringir as alterações do modelo de objetos de modo a mantê-lo ontologicamente bem fundamentado.

[15]Epistemologia é o estudo do conhecimento e da crença justificada [65]. Ontologia é o estudo do ser. Uma pessoa possui necessariamente uma idade. Entretanto, ao representar uma pessoa em sistema, pode-se não conhecer sua idade ou pode-se conhecer a idade errada. A verdadeira idade da pessoa é um aspecto ontológico. O cadastro da idade (em outras palavras, do que se sabe sobre a idade) é um aspecto epistemológico.

Neste sentido, de acordo com a classificação sugerida pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq)[16], esta pesquisa contribui para a área de *Sistemas de Informação*. *Sistemas de Informação* é uma especialização da área de *Metodologia e Técnicas de Computação* que, por sua vez, especializa a área de *Ciência da Computação*.

1.5 Método de Pesquisa

1.5.1 Classificação da pesquisa

Do ponto de vista de sua natureza [66, p. 8], esta é uma *pesquisa aplicada*: pretende gerar conhecimentos para aplicação prática em modelagem de software para solucionar problemas relacionados à construção de software em domínios que exijam frequentes mudanças e evoluções de modelos.

Do ponto de vista do modo de tratamento do problema [66, p. 8 e 9], a pesquisa classifica-se como *qualitativa*, pois nem todos os seus aspectos são quantificáveis.

Quanto à finalidade da pesquisa [66, p. 9], esta pesquisa é *exploratória e descritiva*: exploratória pela necessidade de sondagem e pesquisa bibliográfica para compreensão da área de pesquisa e da área de aplicação da pesquisa; descritiva por expor características da arquitetura e dos modelos apresentados para solução do problema de pesquisa.

Este trabalho utiliza como principais meios de investigação [66, p. 9 e 10]: a *pesquisa bibliográfica* e a *design science research*.

Esta pesquisa busca o aprimoramento de fundamentos científicos relacionados ao estudo de metamodelagem associada à ontologia de fundamentação. A base empírica e teórica utilizada é concentrada nos temas Modelagem conceitual e Ontologia, Lógica Clássica, Lógicas Modais e Engenharia de Software.

O referencial teórico apresentado no Capítulo 2 baseia-se principalmente em modelagem conceitual, metamodelos, modelos adaptativos de objetos, ontologias e ontologias de fundamentação. O levantamento bibliográfico possui base exploratória em material já publicado, como livros, artigos de periódicos, artigos de revistas, anais de eventos, dissertações e teses.

[16] `http://lattes.cnpq.br/web/dgp/ciencias-exatas-e-da-terra`

1.5.2 Aspectos Técnicos do Texto

Tradução dos termos da UFO

O processo de tradução de termos ou textos de uma língua para outra pode induzir a atribuição de significados não desejados aos conceitos descritos pelo autor original. Este fato foi particularmente relevante para a decisão sobre as traduções dos termos da UFO citados nesta dissertação. Alguns trabalhos referenciados neste texto compartilharam óbices semelhantes (Zamborlini [67, p. 29], Araujo [30, p. 69]).

Além disso, cabe ressaltar que, apesar do autor da UFO ser brasileiro, poucos dos textos com autoria ou coautoria de Guizzardi estão em língua portuguesa. Alguns exemplos são [68–71]. Neles, as traduções dos termos não são uniformes. Em alguns casos são traduzidos. Em outros casos, não são.

Para evitar a introdução de novas interpretações, optamos por seguir dois critérios para tradução:

1. Assim como Zamborlini [67], as figuras da UFO são mantidas na língua inglesa, para evitar a introdução de novos significados aos seus conceitos;

2. Entretanto, no corpo do texto, os termos são traduzidos de acordo com o *Glossário da UFO* da tese de Araujo [30, p. 437-500]. Este glossário provavelmente é aquele com a cobertura bibliográfica mais ampla da UFO em língua portuguesa. O levantamento bibliográfico necessário para a produção da tese cobre "um espaço temporal de cerca de 10 anos de publicações sobre a ontologia, contados a partir de 2005. No caso, investiga-se especialmente os trabalhos publicados pelo autor da UFO, individualmente ou em colaboração." [30, p. 66].

1.6 Estrutura do Manuscrito

Esta dissertação está organizada da seguinte maneira:

- **Capítulo 2:** apresenta o embasamento teórico necessário para o entendimento deste trabalho;
- **Capítulo 3:** propõe um AOM para modelos conceituais que implementem a UFO.
- **Capítulo 4:** apresenta um exemplo de inclusão de um modelo de objetos no modelo de aplicação;
- **Capítulo 5:** conclui este trabalho e apresenta os trabalhos futuros.

Capítulo 2

Embasamento Teórico

Neste Capítulo, são abordados e definidos os conceitos necessários para o embasamento deste trabalho. Na Seção 2.1, apresentam-se conceitos importantes sobre modelos, metamodelos, modelagem conceitual e modelos adaptativos de objetos. A Seção 2.2 trata de elementos básicos sobre ontologias e Ontologia (com O maiúsculo) e faz uma comparação sobre o significado da palavra ontologia na Ciência da Computação, na Ciência da Informação e na Filosofia. A Seção 2.3 e a Seção 2.4 descrevem, respectivamente, o quadrado ontológico aristotélico e o dodecágono ontológico, principais inspirações para o modelo descrito no Capítulo 3. A Seção 2.5 descreve a UFO, ontologia que possui as restrições necessárias para a construção dos modelos conceituais suportados pelo modelo lógico proposto.

2.1 Modelos, Metamodelos e Modelagem Conceitual

Os métodos utilizados na Engenharia de Software para a construção de aplicações visam representar a realidade por meio de modelos computacionais interpretados por computadores. A construção de modelos utiliza-se de abstrações que, por sua vez, são resultado (ou o processo) de análise de entidades pelas características relevantes observadas por determinado ponto de vista [5, p. 10].

Neste sentido, modelos são representações simplificadas de um original. Eles são construídos pela abstração dos principais conceitos seguidos de posteriores representações e transformações [6, p. 3]. Logo, modelos são abstrações coerentes de elementos formais que descrevem parte de uma visão de mundo para alguma finalidade passível de posterior análise [72].

De acordo com Stachowiak [73], modelos possuem três características:

- São sempre representações dos seus originais;

- Possuem apenas parte dos atributos que os originais dispõem: apenas aqueles relevantes para a perspectiva dos criadores do modelo;
- Substituem os originais para alguma finalidade específica.

A construção de software envolve a produção de diferentes modelos. Modelos de fases preliminares da produção de software tendem a ser mais abstratos que os demais. Um processo de software robusto deve gerar modelos precisos, representando o domínio do sistema em diferentes graus de abstração. Deve também permitir a transformação de modelos mais abstratos em modelos mais detalhados e concretos [6, p. xiii, xiv].

Modelagem é processo essencial para a construção de software de qualidade. Modelar envolve entender e representar um problema. Para a construção de uma solução de software adequada, primeiro faz-se importante conhecer o problema a ser solucionado, ou seja, convém abordar aspectos mais próximos ao espaço do problema do que ao espaço da solução. Logo, entender o que um sistema deve fazer é, naturalmente, necessário para possibilitar sua construção. Por isso, esquemas conceituais e modelos de domínio são essenciais para a codificação de um sistema de informação [6, p. 17].

De acordo com Aßmann et al. [74, p. 249], um processo de desenvolvimento de software baseado em modelos passa por três fases:

1. *análise*: fase em que se faz levantamento de requisitos e se constrói o *modelo de domínio*, *modelo de negócios* e *modelo de contexto*;
2. *projeto*: fase em que se produz a especificação da arquitetura e do projeto do software;
3. *implementação*: em que o projeto de software é implementado.

Conforme apresentado na Subseção 1.2.1, vários estudos apontam que as atividades de análise são as mais críticas para o processo de desenvolvimento de sistemas de informação. Erros encontrados nessas fases têm alto impacto na qualidade, confiança e nos custos dos produtos de software. Pastor e Molina [6, tradução nossa] afirmam: "Para descrever algo corretamente, é necessário um conjunto de conceitos dos quais a semântica é definida sem ambiguidades" [1].

É possível converter modelos mais abstratos em modelos mais concretos por meio de transformações manuais, automáticas ou semiautomáticas. Para as transformações automáticas ou semiautomáticas, os modelos precisam estar conectados de modo que elementos modelados sejam rastreados do modelo mais abstrato para outro mais concreto e vice-versa [74, p. 250].

[1] *In order to correctly describe something, there must be an adequate set of concepts of which the semantics are defined unambiguously.*

Ainda neste sentido, modelos estão relacionados à realidade. De acordo com Pidd [75, p. 12, tradução nossa], "um modelo é uma representação explícita e externa de parte da realidade pela visão das pessoas que desejam usá-lo para entender, modificar, gerenciar e controlar aquela parte da realidade"[2]. O relacionamento com a realidade divide modelos em dois grupos: modelos descritivos e modelos prescritivos [74]:

1. *Modelos descritivos* são aqueles que descrevem a realidade. Neste caso, os modelos são derivados da realidade, são representações desta. A realidade é independente do modelo e este é simplificado, incompleto, pois somente os aspectos da realidade relevantes para o propósito do modelo serão representados. O que não está no modelo é desconhecido, mas pode existir.

2. *Modelos prescritivos*, por sua vez, definem a realidade. Ou seja, as estruturas e os comportamentos da realidade são criados de acordo com o modelo. Neste caso, o modelo é completo. Toda informação para a construção da realidade já está nele.

Geralmente, modelos de software são prescritivos. O software é a realidade construída com base no modelo. Assim, o modelo é a especificação do software. Portanto, independente da relação do modelo com a realidade ser descritiva ou prescritiva, modelos sempre a representam. Como representam a realidade, modelos devem ser fiéis a ela, ou seja, qualquer conclusão obtida pela análise do modelo deve levar a mesma conclusão sobre a realidade.

Além disso, modelos podem descrever tanto estruturas quanto comportamentos. Os modelos estruturais são aqueles que descrevem conceitos da realidade e as relações entre esses conceitos (semântica estática de um domínio, estrutura livre de contexto). Modelos comportamentais especificam aspectos comportamentais dos conceitos e relacionamentos abstraídos [74].

2.1.1 Metamodelos

Um modo comum de emprego de modelos é utilizar propriedades compartilhadas por indivíduos de um domínio para agrupá-los em diferentes classes. Utilizar-se de metapropriedades compartilhas por classes para se criar metaclasses (classes de classes) resulta na produção de "modelos de modelos", chamados de metamodelos [76].

Um metamodelo é um modelo de uma linguagem de modelagem. Por isso, ele faz declarações sobre o que pode ser expresso em modelos válidos da linguagem de modelagem e permitem especificar classes de sistemas em que o próprio sistema é um modelo válido expresso na linguagem [77, p. 28 e 29].

[2] *A model is an external and explicit representation of part of reality as seen by the people who wish to use that model to understand, to change, to manage and to control that part of reality*

2.1.2 Modelo de Objetos e Modelo Adaptativo de Objetos

Um *Modelo de Objetos* é, na visão de Jones [78], tanto um conceito quanto uma ferramenta. Ele possui diretrizes para descrever entidades abstratas em termos do pensamento humano. O uso do *Modelo de Objetos* permite esclarecer e tornar explícitas as relações de dependência entre as entidades do domínio modelado. Sua ênfase é caracterizar os componentes do mundo físico ou abstrato que serão modelados como um software. Rumbaugh et al. [79, p. 6] definem modelo de objeto como a estrutura estática dos objetos de um sistema e seus relacionamentos. Já para Cattell e Barry [80, p. 12, tradução nossa], "a semântica de um modelo de objetos determina as características de objetos, como objetos são relacionados uns com os outros e como objetos podem receber nomes e serem identificados"[3].

Algumas arquiteturas possuem um *Modelo Adaptativo de Objetos* (*Adaptive Object Model* (AOM))[4], um *tipo de metamodelo* que permite que categorias de objetos sejam modificadas em tempo de execução e oferece mais flexibilidade e adaptabilidade a um sistema [19, 81].

Neste estilo arquitetural, há dois *modelos de objetos*: um *modelo de objetos de domínio*, o qual é armazenado em um banco de dados e possui as categorias e os relacionamentos do domínio modelado; e um *modelo de objetos da aplicação* que funciona como um interpretador do modelo de objetos de domínio [19]. Em tempo de execução, categorias e relacionamentos entre categorias podem ser acrescentados ou modificados no modelo de objetos de domínio sem afetar o modelo de objetos da aplicação.

Arquiteturas AOM diferem de projetos orientados a objetos tradicionais. Estes geram classes para as diferentes categorias do domínio. Os relacionamentos entre classes são implementados por meio de atributos e métodos. Já nos AOMs, as categorias do domínio são instâncias do metamodelo. Dessa forma, alterações de modelagem implicam em alteração de código nas abordagens tradicionais de orientação a objetos e, consequentemente, em uma nova versão de software; em sistemas AOMs, as mesmas alterações de modelagem podem ser realizadas diretamente no banco de dados, em tempo de execução, sem necessidade de alteração de código-fonte e, consequentemente, de lançamento de uma nova versão de software.

Yoder et al. [19] resumem as principais características de sistemas AOM do seguinte modo:

- representam classes, atributos e relacionamentos como metadados;

[3] *the semantics of the Object Model determine the characteristics of objects, how objects can be related to each other, and how objects can be named and identified.*

[4] também conhecido por *Object System*, *Runtime Domain Model*, *Active Object Model*, *Dynamic Object Model* [81]; *Knowledge Level* [19, 82]

- possuem modelos baseados em instâncias e não em classes;
- para modificar o comportamento do sistema, os usuários alteram o *modelo de objetos* para refletir mudanças de representação domínio;
- o *modelo de objetos* é gravado no banco de dados e é interpretado;
- o *modelo de objetos* é ativo, ou seja, o sistema reage imediatamente às alterações do *modelo de objetos*.

Em AOMs, as entidades do domínio "são modeladas por meio de descrições que são interpretadas em tempo de execução." [19, tradução nossa, p. 2].

Para Yoder et al. [19, p. 4], o projeto de AOM envolve três atividades:

- descrever entidades, regras e relacionamentos do domínio;
- projetar um mecanismo para instanciar e manipular as entidades do domínio conforme suas regras na aplicação;
- desenvolver ferramentas para descrever estas entidades, regras e relacionamentos.

A principal vantagem de um AOM é a flexibilidade e a facilidade para alterar o modelo de objetos. Esta abordagem é adequada para sistemas cujos modelos sejam alterados com frequência ou naqueles em que se deseja permitir aos usuários alterar, configurar ou extender o sistema dinamicamente. Também é adequada para definições incompletas, mas iterativas, do domínio. O código de um AOM tende a ser muito menor que o código de um sistema tradicional, uma vez que o modelo de aplicação é realizado através de um metamodelo. Já as tabelas de banco de dados tendem a ter muito mais tuplas, já que o *modelo de objetos* é gravado em banco de dados. Como as informações que estariam em código passam a ser registradas em banco, alterações na especificação de software não necessariamente implicam em alteração do código-fonte ou em propriedades de tabelas de banco de dados [19].

Para Riehle et al. [81], AOM são úteis nos seguintes casos:

- o domínio possui muitas categorias de objetos que diferem apenas em algumas propriedades;
- deseja-se criar novas categorias de objetos com diferentes propriedades em tempo de execução;
- o usuário final pode alterar categorias de objeto;
- alterações de modelagem são frequentes ou o sistema exige evolução rápida;

- há necessidade de um modelo explícito da estrutura dos objetos e esse modelo estrutural necessita ser acessado em tempo de execução;
- pretende-se permitir validação automática das categorias do domínio;
- para construir uma linguagem de modelagem específica de domínio.

Em contrapartida, Yoder et al. [19] argumentam que projetos com AOM podem apresentar graves lacunas entre arquitetos e desenvolvedores, as quais envolvem desde aspectos relacionados a ponto de vista (projeto vs. implementação) a aspectos concernentes ao entendimento de abstrações, dos elementos do domínio do problema etc. Estas lacunas seriam, em grande parte, relacionadas aos vários níveis de abstração existentes em sistemas AOM. Vários níveis de abstração implicam na existência de vários pontos a serem modificados. Esta seria a causa do distanciamento conceitual entre arquitetos e desenvolvedores.

Sistemas AOM frequentemente possuem complexidade adicional para a construção de um mecanismo extra para interpretação do *modelo de objetos*. Programadores, neste caso, constroem um mecanismo para execução do modelo e não o modelo em si. Ou seja, há uma desconexão entre o modelo de sistema e seu comportamento, [18]. Também é mais complexo definir os relacionamentos e atributos de entidades.

O custo inicial de desenvolvimento de sistemas AOM pode ser mais alto quando comparado ao desenvolvimento de um sistema tradicional. Eles também são mais difíceis de construir e de entender, visto que há dois sistemas coexistindo: o interpretador construído de forma tradicional e o AOM. Além disso, classes não representam abstrações do domínio, visto que informações do domínio estão em banco [19]

Entretanto, este custo extra traz como vantagem a possibilidade de alteração da modelagem em tempo de execução. Em uma abordagem orientada a objetos tradicional, criam-se classes para as entidades do domínio e os relacionamentos com outras entidades são modelados por meio de atributos nas classes. Ações executadas pelas classes são codificadas por meio de métodos. Ou seja, neste caso, uma mudança na modelagem das entidades do domínio implica em alteração de código e criação de uma nova versão de software.

Em suma, sistemas AOM oferecem mais flexibilidade e velocidade de alteração de software. Em contrapartida, eles tendem ser mais difíceis de entender e de manter.

2.2 Ontologias

Ontologia (com O maiúsculo), na Filosofia, é o estudo do ser [83]. É a parte da Metafísica que trata da existência, da natureza e dos relacionamentos entre os seres. Ao

longo do tempo, o termo ontologia foi ganhando significados distintos. Em Computação, ontologias são definidas de duas formas: em modelagem conceitual e áreas correlatas, seu significado está mais próximo daquele da Filosofia, ou seja, "um sistema de categorias formais independentes de domínio e filosoficamente bem fundamentado que pode ser usado para enunciar modelos da realidade específicos de domínio" [68]; por outro lado, em áreas como Inteligência Artificial e Web Semântica, o termo ontologia tem sido usado para se referir a uma estrutura de conceitos com vocabulário lógico usada para representação de um domínio específico [84].

No sentido filosófico, Lowe [85] defende que apesar da Ontologia ser uma ciência *a priori* – uma ciência independente de todas as experiências particulares, ou seja, uma ciência não empírica –, ela admite elementos empíricos. O autor advoga que a parte *a priori* da Ontologia explora a possibilidade metafísica para tentar estabelecer quais *categorias* de seres podem existir e coexistir para formar um mundo possível único. Já a parte empírica, trata de explorar quais *categorias* de seres podem existir neste mundo.

Para Gruber [86], uma ontologia é a especificação explícita de uma visão de mundo simplificada, abstrata, que é representada para alguma finalidade. O conjunto de objetos do domínio representado, também chamado de universo de discurso, e as relações entre esses objetos comumente são representados por meio de termos que os descrevem. Portanto, a ontologia de um domínio pode ser descrita por meio de termos que representam os objetos do domínio.

De acordo com Gruber [86, tradução nossa]:

> Nesta ontologia, definições associam os nomes de entidades do sistema no universo do discurso (por exemplo, classes, relações, funções, ou outros objetos) com texto legível para humanos descrevendo o que os nomes devem significar, e axiomas formais que restringem a interpretação e permitem o uso correto desses termos.[5]

De acordo com Aßmann et al. [74, p. 255, tradução nossa], "uma ontologia é um modelo compartilhado por um grupo de pessoas em um determinado domínio"[6]. Segundo esta visão, uma ontologia é conhecimento compartilhado que precisa ser padronizado para a comunicação entre membros de um grupo de pessoas.

O termo ontologia também é utilizado em ciência da informação. Almeida [84] apresenta um estudo breve sobre os diferentes usos do termo ontologia na Filosofia, na Ciência da Computação e na Ciência da Informação. A Tabela 2.1 sumariza a análise realizada.

[5] In such an ontology, definitions associate the names of entities in the universe of discourse (e.g., classes, relations, functions, or other objects) with human-readable text describing what the names are meant to denote, and formal axioms that constrain the interpretation and well-formed use of these terms.

[6] an ontology is a model shared by a group of people in a certain domain.

Tabela 2.1: Resumo das visões sobre Ontologia e ontologias [84].

Distinção	**Campo**	**O que é?**	**Propósito**	**Exemplo**
Ontologia como uma disciplina	Filosofia	Ontologia como um sistema de categorias	Entender a realidade, as coisas que existem e suas características	Sistemas de Aristóteles, Kant, Husserl
Ontologia como um artefato	Ciência da Computação	ontologia como uma teoria (baseada em lógica)	Entender um domínio e reduzi-lo à modelos	BFO, DOLCE (genéricas)
		ontologia como um artefato de software	Criar um vocabulário para representação em sistemas e para gerar inferências	OWL (linguagem de RC)
	Ciência da Informação	ontologia como uma teoria (informal)	Entender um domínio e classificar termos	Sistema de classificação de Ranganathan
		ontologia como um sistema conceitual informal	Criar vocabulários controlados para recuperação da informação a partir de documentos	um catálogo, um glossário, um tesauro

2.3 Quadrado Ontológico Aristotélico

Na Seção 2.2, viu-se que a Ontologia estuda quais categorias de seres existem. Essas categorias ontológicas são organizadas hierarquicamente. Elas são identificadas e individualizadas pelas características de identidade dos seus membros. Nesta organização hierárquica, o nível mais alto da hierarquia contém a categoria mais elementar: *ser*, *coisa* ou *entidade*. Lowe [85] posiciona, no segundo nível mais alto/abstrato, duas categorias: a categoria dos *Universais* e a categoria dos *Particulares*[7].

Na Subseção 1.2.3, discute-se brevemente sobre o problema dos Universais e dos Particulares, o qual é essencial para a Modelagem Conceitual e para a Engenharia de Software. A ontologia de quatro categorias presente em Lowe [85] trata deste problema. Para entendê-lo, convém explicar três conceitos: *objeto*, *universal* e *moment*.

Um objeto (ou particular) é um indivíduo que possui identidade própria, ou seja, ele existe independentemente de outros indivíduos. Objetos carregam propriedades. Um uni-

[7]Uma proposta diferente é descrita por Chisholm [87]

versal (pelo menos de primeira ordem) é um padrão que repete em diferentes particulares em espaço e tempos distintos. *Moments* são propriedades particulares dependentes de um e somente um objeto [85].

Existem posições ontológicas conflitantes quanto a duas questões:

1. Qual dessas três categorias existem?
2. Quais delas são fundamentais?

A posição de Lowe [85] é de que as três categorias existem e as três são fundamentais. Entretanto, o autor propõe uma variante deste posicionamento: as três categorias existem, mas os universais são divididos em dois grupos: *Universais* cujas instâncias são objetos e *Universais* cujas instâncias são *moments*. Universais cujas instâncias são objetos são *espécies* de objeto; universais cujas instâncias são *moments* são *propriedades* de objetos. Desta forma, Lowe propõe a ontologia de quatro categorias ontológicas fundamentais, conforme apresentado na Figura 2.1. Além das quatro categorias ontológicas, a Figura 2.1 apresenta três tipos de relacionamentos entre elas.

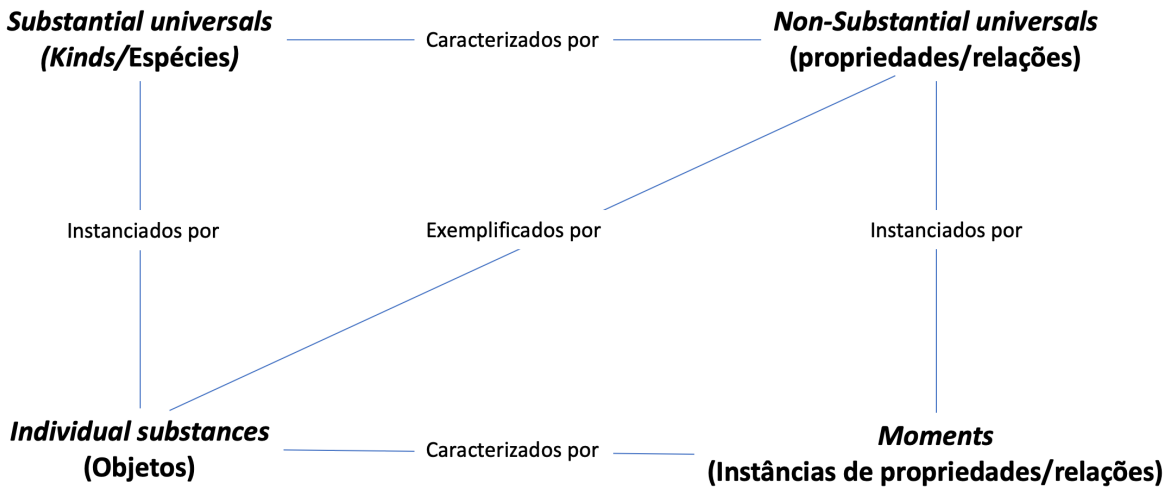


Figura 2.1: Quadrado ontológico / Ontologia de quatro categorias. Adaptado de Lowe [85]

Para Lowe, uma categoria ontológica é um *tipo de ser*[8]. Em outras palavras, "categorias ontológicas são tipos de entidades que são definidas por condições de existência e de identidade cuja natureza é definida *a priori*" [85, p. 20, tradução nossa]. A mais elementar das quatro categorias é a de *substâncias individuais* ou *objetos*. Para cada objeto existe um universal correspondente de modo que o *objeto* é instância de uma *Espécie*

[8] *ser*, no sentido desta expressão, é verbo. Não é substantivo.

(Kind) universal. A outra categoria de universais, *Non-substantial universals*, abrange *propriedades* e *relações*. Por fim, a quarta categoria contém as instâncias dessas propriedades/relações – chamadas por Lowe de *modes*, mas também conhecidas por *moments*, acidentes individuais, qualidades particulares.

Estas categorias se relacionam por meio de três tipos de relacionamentos: *instanciado por*, *caracterizado por* e *exemplificado por*. De acordo com Merriam-Webster [88], *instanciar* significa representar (uma abstração) por meio de uma instância concreta. Instanciação é o relacionamento entre um *objeto* e sua respectiva *espécie/kind* ou entre um *moment/mode* e seu respectivo universal. Por exemplo, uma bola em particular instancia a *espécie* bola. O relacionamento entre as duas categorias de universais e o relacionamento entre as duas categorias de particulares são chamados de caracterização. A *espécie* é caracterizada pelo universal não substanciado. Do mesmo modo, um objeto é caracterizado por um *mode* particular. Por último, Lowe [85] sugere que o relacionamento entre um objeto e seus universais não substanciados seja chamado de exemplificação, de modo que um objeto individual exemplifica um universal ligado indiretamente ao objeto por meio de um *mode* que caracteriza o objeto e seja instância desse universal.

2.4 Dodecágono Ontológico

Para atividades de modelagem conceitual, Guizzardi e Wagner [89] defendem a necessidade de doze categorias para atividades de modelagem conceitual, ou seja, o quadrado ontológico não seria suficiente. As doze categorias são representadas na Figura 2.2. O dodecágono é uma extensão da ontologia base da OWL, conforme apresentado na 2.3.

Tal como ocorre na ontologia da OWL, Guizzardi e Wagner sugerem a divisão de *Non-substantial universals* em duas categorias de *tipos de relacionamentos binários*:

1. *reference properties*, cujo contradomínio são *Substantial Universals*;
2. *attributes*, cujo contradomínio são *Datatypes*.

Um *Datatype*, por sua vez, representa os possíveis valores de um atributo e não é representado no quadrado ontológico. *Datatypes* representam valores tais como inteiros, números decimais, datas, cadeias de caracteres etc.

Assim, a ontologia de oito categorias da OWL, portanto, possui quatro categorias de universais: *Data type*, *attributes*, *object type* e *reference properties (ou Binary relationship type)*. As instâncias desses quatro tipos de universais completam as oito categorias ontológicas dessa ontologia.

Entretanto, Guizzardi e Wagner [89] argumentam que essas oito categorias não são ricas o suficiente para representar todas as categorias fundamentais de indivíduos necessá-

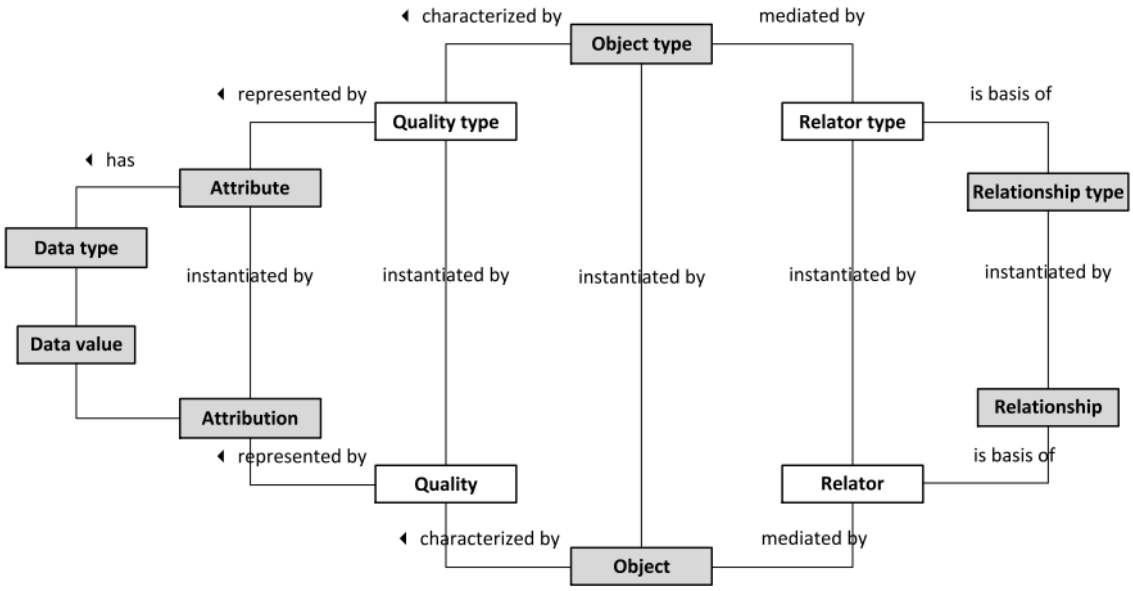


Figura 2.2: Dodecágono Ontológico proposto por Guizzardi e Wagner [89]

rios para modelagem conceitual. Portanto, sugerem a necessidade de mais duas categorias de indivíduos (*qualities* e *relators*), assim como das duas categorias de universais que instanciam.

Qualities, segundo os autores, são representados por *attributions*. Por exemplo, a cor de um gato é um *quality*. É algo inerente ao gato e somente àquele gato. A partir da afirmação "A cor de *Snowbell* é branca", desprende-se que o *truthmaker* desta afirmação é a *attribution* que associa o valor de uma cor ao objeto representado pelo nome *Snowbell*. Portanto, essa *attribution* representa o *quality*.

Do mesmo modo, *relationships* representam *relators* [89]. Como os autores atestam, é possível que diferentes *relationships* representem o mesmo *relator*. Por exemplo, *Maria é casada com João* e *João é casado com Maria* são dois relacionamentos que representam o mesmo *relator*.

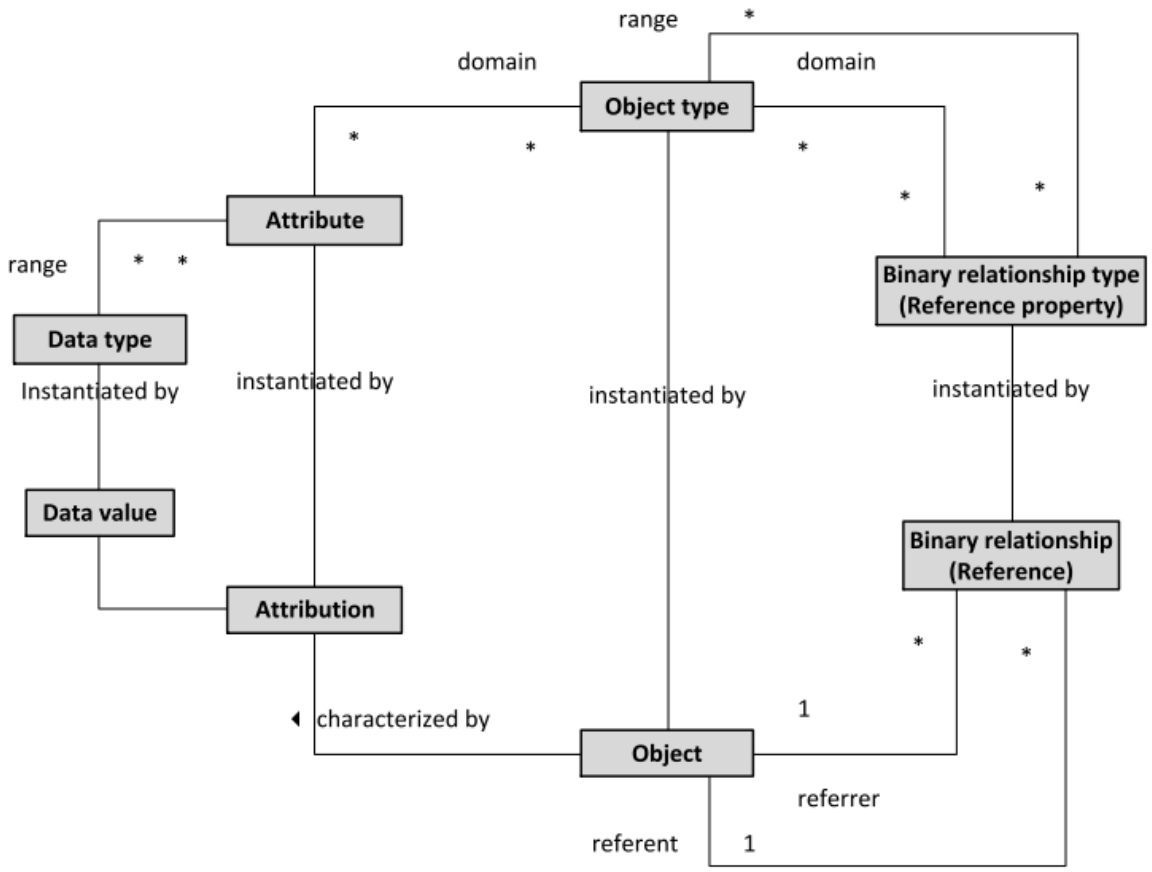


Figura 2.3: Ontologia de oito categorias da OWL. Fonte: Guizzardi e Wagner [89]

2.5 Ontologia de Fundamentação UFO

Nesta Seção, apresentam-se conceitos gerais sobre a UFO. Aspectos dos fragmentos da UFO-A e UFO-B utilizados no desenvolvimento deste trabalho são detalhados na Seção 2.6 e na Seção 2.7. Não é objetivo deste manuscrito ser uma referência abrangente da UFO. Para isto, a referência inicial mais indicada é a tese de doutorado de Guizzardi [32]. Uma versão mais concisa pode ser encontrada em Guizzardi [90]. Os dois trabalhos foram publicados em língua inglesa. Para referência em língua portuguesa, algumas opções são os trabalhos de Zamborlini [67] e Araujo [30]. A UFO continua em desenvolvimento por meio de diversas publicações. Ainda não existe obra única que alberga todos estes elementos. Por isso, as obras citadas abrangem apenas parte desta ontologia de fundamentação. Esta Seção apontará algumas referências úteis para a compreensão de tópicos específicos da ontologia.

Os principais fragmentos da UFO são: UFO-A – Ontologia de Objetos (Endurantes),

UFO-B – Ontologia de Eventos (Perdurantes) e UFO-C – Ontologia de Entidades Sociais. A UFO-A é uma ontologia de endurantes, seres que existem no espaço, independente do tempo observado. Ela trata aspectos de estrutura na modelagem conceitual. A UFO-B é uma ontologia de perdurantes (eventos e processos) e trata de seres com ocorrência determinada no tempo. A UFO-C, por sua vez, é uma ontologia de entidades sociais, construída e embasada na UFO-A e na UFO-B. Ela trata de conceitos como crenças, desejos, intenções, dentre outros [60]. As principais características dos três fragmentos citados são sumariadas abaixo:

1. UFO-A: ontologia de endurantes que trata de aspectos de modelagem conceitual de estruturas. É organizada como uma ontologia de quatro categorias, baseada no trabalho de Lowe [85] (Seção 2.3). A UFO-A engloba teorias de estruturas de tipos e taxonômicas [50, 51] conectada a uma teoria de identificadores de objetos (incluindo uma semântica formal em lógica formal quantificada de sortais [52]), relações todo-parte [53], [54], propriedades intrinsecamente particularizadas, atributos e espaços de valores de atributos [55, 56], relações e propriedades relacionais particulares [57, 58] e papéis [59];

2. UFO-B: ontologia de perdurantes (eventos e processos) que trata de aspectos tais como mereologia de perdurantes, ordem temporal de perdurantes, participação de objetos em perdurantes, causalidade, mudança e a conexão entre perdurantes e endurantes por meio de disposições [91];

3. UFO-C: ontologia de entidades sociais, fundamentada nas UFO-A e UFO-B. A UFO-C trata de noções como crenças, desejos, intenções, objetivos, ações, compromissos e reivindicações, papeis sociais e *relators* sociais, dentre outros [92, 93].

Além de ser uma ontologia de fundamentação cujo núcleo baseia-se na ontologia de quatro categorias, a UFO foi construída e fundamentada em teorias de áreas como ontologia formal na filosofia, ciência cognitiva, lógica filosófica e linguística, e desenvolvida para fornecer fundamentação ontológica para linguagens gerais de modelagem conceitual. Ela começou como a unificação de duas ontologias: a *Generalized Formalized Ontology* (GFO)/*General ontological language* (GOL) [39, 40, 94] e a Ontoclean[9]/DOLCE [44]. Apesar de importantes, essas ontologias possuem problemas relacionados ao desenvolvimento de fundamentos ontológicos para linguagens de modelagem conceitual de uso geral, como *Enhanced Entity–Relationship* (EER), *Unified Modeling Language* (UML), *Object-Relational Mapping* (ORM). A UFO, por sua vez, unifica essas ontologias de modo a superar as limitações detectadas e utilizando-se de seus pontos positivos [32, 60, 95].

[9] `http://www.ontoclean.org`

Para classificar e definir suas categorias, a UFO utiliza-se de algumas meta-propriedades que diferenciam as categorias de seres.

Três dessas meta-propriedades são [67]:

- *identidade*: alguns conceitos compartilham características que os permitem serem identificados e contados por serem mais ou menos iguais. Por exemplo, um *Clydesdale* e um *Puro Sangue Inglês* compartilham a propriedade de serem *cavalos*. Esses dois indivíduos podem ser classificados e contados de acordo com este critério, ou seja, temos dois cavalos nesta amostra. Entretanto, os mesmos cavalos poderiam compartilhar a propriedade de serem *bonitos*. Bonito não fornece um critério de identidade: não faz sentido dizer que temos dois bonitos neste estábulo;

- *rigidez*: quanto a rigidez, os designadores de conceitos podem ser *rígidos*, *anti-rígidos* e *semi-rígidos*. Um conceito é rígido se ele define uma coisa em todos os mundos possíveis onde aquela coisa existe e não define nada além disso [96]. *Cavalo* é rígido se todos os particulares por ele definidos, tais como *Baloubet du Rouet*, *Man O' War* etc, tanto no mundo atual como em um mundo contrafactual, sempre forem um *Cavalo*. Um conceito é anti-rígido se aplicado aos indivíduos de modo circunstancial. O conceito *Empregado* é anti-rígido pois ele define particulares que podem deixar de ser empregados, mas continuarem a existir. Um conceito é semi-rígido se for imperiosamente aplicado a alguns particulares e ocasionalmente para outros [67].

- *dependência*: um *universal A* é relacionalmente dependente de um *universal B*, se todos os particulares que instanciam A precisam necessariamente participar de uma relação com um particular que instancia o universal B. Zamborlini [67] apresenta três tipos de dependência:

  - *Dependência genérica*: é aquela em que a relação que configura a dependência pode sofrer alterações. Por exemplo, uma *roda* é dependente genericamente de um *automóvel*, se roda sempre for parte de um automóvel, mas podendo ser parte de automóveis diferentes em momentos diferentes.
  - *Dependência específica*: é aquela em que a relação que configura a dependência é imutável. O *cérebro* depende especificamente de *ser vivo*, pois não é possível transplantar um cérebro de um indivíduo para outro.
  - *Dependência existencial*: é uma dependência específica em que os particulares dependentes só existem se os particulares de quem eles dependem existirem.

O mesmo *cérebro* é dependente existencialmente de *ser vivo*, caso aquele não possa existir sem este[10].

Nas Seções seguintes, apresentam-se aspectos importantes da UFO-A e da UFO-B para o desenvolvimento dessa proposta. Outros fragmentos da UFO – tais como a UFO-C, UFO-L, UFO-S – estão fora do escopo deste trabalho.

2.6 UFO-A

De acordo com Guizzardi et al. [68], a UFO-A é uma ontologia de *endurantes* que define o núcleo da UFO, conforme apresentado na Figura 2.4. Ela abrange teorias de estruturas de tipos e estruturas taxonômicas [50, 51], uma teoria de identificadores de objetos (incluindo uma semântica formal em lógica formal quantificada de sortais [52]), relações todo-parte [53, 54], propriedades intrinsecamente particularizadas, atributos e espaços de valores de atributos [55, 56], relações e propriedades relacionais particulares [57, 58] e papéis [59].

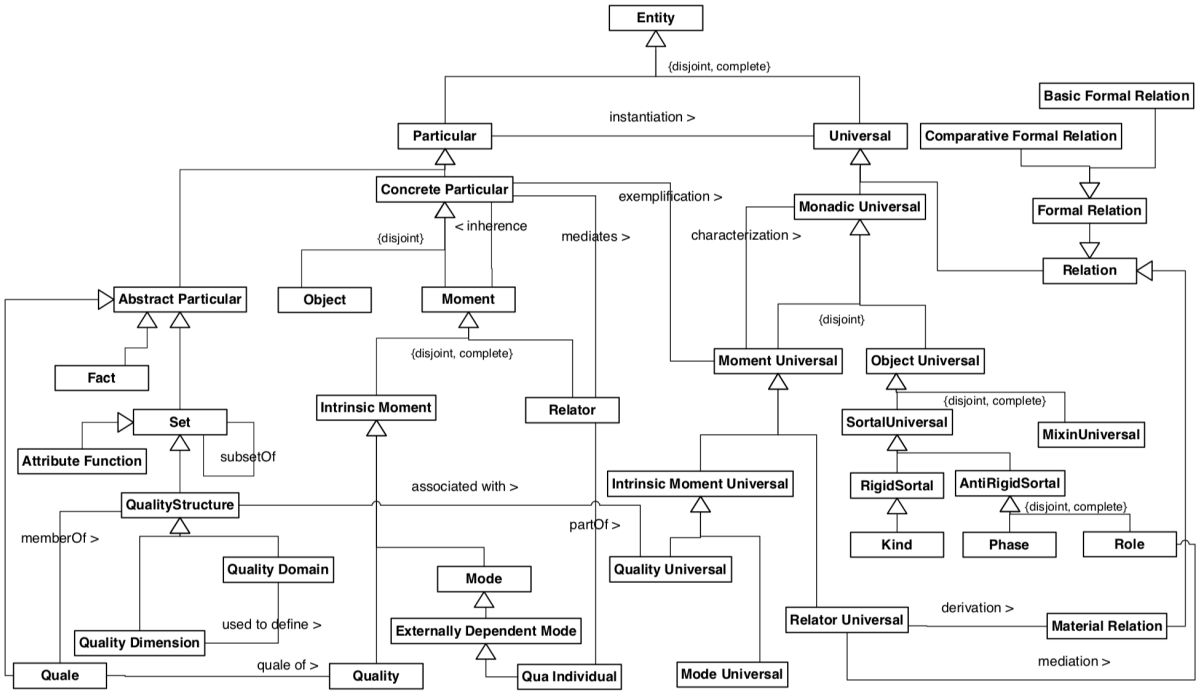


Figura 2.4: Fragmento da UFO-A. Fonte: Guizzardi e Wagner [95]

[10]Existem tipos de dependências específicas que não são existenciais. Por exemplo, a dependência histórica, na qual uma entidade α depende de uma entidade β para iniciar sua existência, mas α, uma vez criada, pode viver de forma independente de β [97, p. 31-32].

A UFO pode ser vista como uma teoria sobre categorias de **universais** e **indivíduos** [67, p. 29], isto é, a ontologia em apreço trata do problema dos universais e particulares mencionado na Subseção 1.2.3.

Conforme apresentado na Seção 2.3, Lowe [85] defende que categorias ontológicas são organizadas hierarquicamente. Nesta hierarquia, a categoria de mais alto nível normalmente é chamada de *ser*, *coisa* ou *entidade*. Lowe [85] divide essa categoria em duas outras, *universais* e *particulares* (ou *indivíduos*), conforme apresentado na Figura 2.5. Guizzardi [32] desenvolveu a UFO-A seguindo o mesmo padrão hierárquico (vide Figura 2.6), em que a categoria mais alta é chamada de *Coisa (Thing)*. *Coisa* é especializada por *Indivíduo (Individual)* e *Universal*.

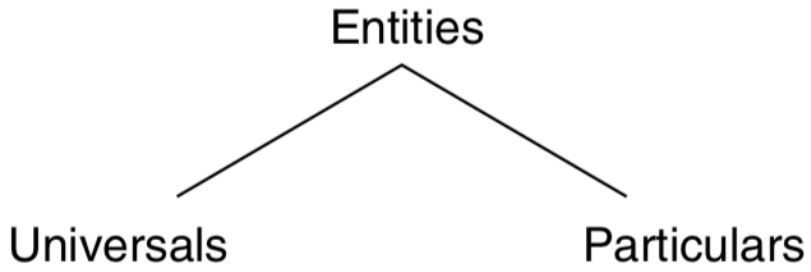


Figura 2.5: Fragmento da hierarquia de categorias proposta por Lowe [85]

Na Figura 2.6, percebe-se que universais são divididos em duas categorias disjuntas: *Monádicos* e *Relações*. Os *Monádicos* (*Monadic Universal*) são a categoria de universais cujas instâncias são indivíduos singulares. As *Relações* (*Relation Universal*) são os universais cujas instâncias são relações.

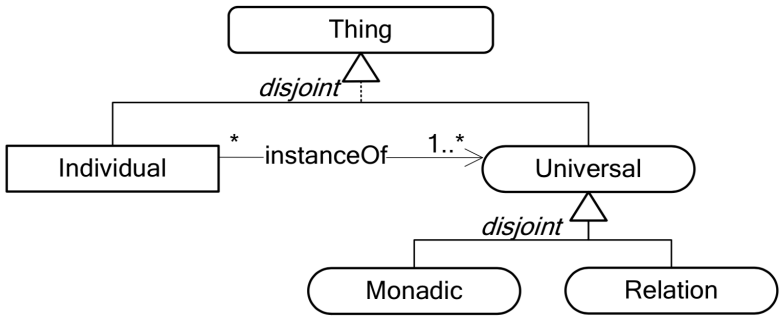


Figura 2.6: Fragmento da UFO - categorias de Universais e Particulares (Indivíduos). Fonte: Zamborlini [67, p. 29]

2.6.1 Indivíduos, Categoria de Indivíduos e Categoria de Universais Monádicos

Zamborlini [67] utiliza-se de um esquema com três níveis (nível meta-conceitual, nível conceitual e nível de indivíduos) para exemplificar o relacionamento entre entidades da UFO com entidades do domínio modelado. A Figura 2.7 exemplifica essa notação. Como é possível observar na Figura, a UFO possui entidades no nível meta-conceitual e no nível conceitual (As entidades da UFO estão destacadas com fundo cinza). Já as entidades do domínio encontram-se nos níveis conceitual e de indivíduos (fundo branco).

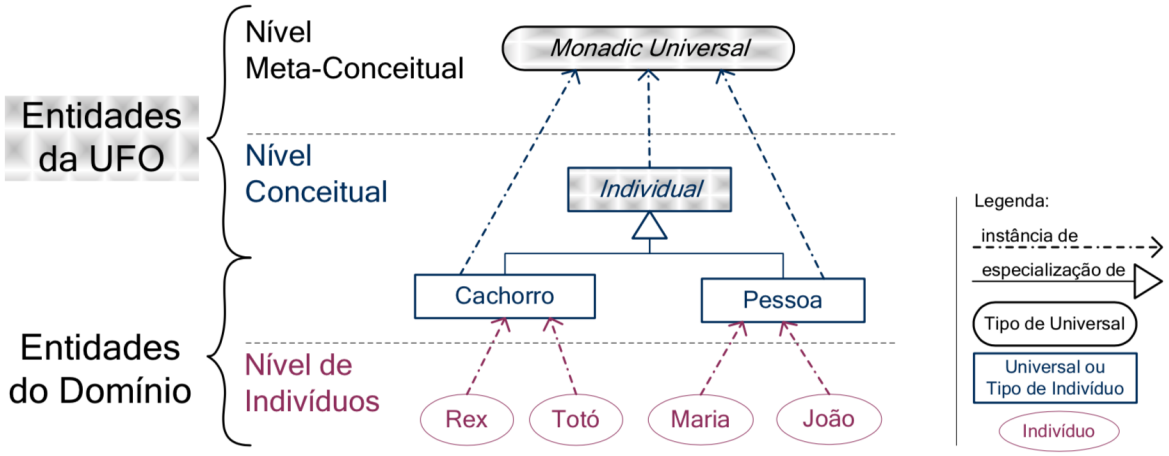


Figura 2.7: Fragmento da UFO em esquema de níveis - Universais Monádicos e Indivíduos. Fonte: Zamborlini [67, p. 31]

Observa-se na Figura 2.7 que o nível meta-conceitual contém a categoria dos *Universais Monádicos*. O nível conceitual contém as categorias presentes no domínio, em que a categoria mais genérica é *Indivíduo*, presente na UFO, e mais dois conceitos de domínio: *Cachorro* e *Pessoa*. O nível de indivíduos contém os próprios indivíduos: *Rex* e *Totó*; *Maria* e *João*. Como o relacionamento de instanciação propaga-se pelo relacionamento de especialização, *Maria* e *João* são instâncias de *Pessoa*, mas também de *Indivíduo*. Do mesmo modo, *Rex* e *Totó* são instâncias de *Cachorro* e instâncias de *Indivíduo*. A Figura 2.7 apresenta dois tipos de relacionamentos básicos:

1. o relacionamento de *instanciação*;
2. o relacionamento de *especialização*.

O relacionamento de instanciação implica em uma entidade ser exemplo de uma categoria do nível imediatamente superior [98]. Por exemplo, *Indivíduo*, *Cachorro* e *Pessoa*

são instâncias de *Universal Monádico*. Do mesmo modo, *Lei Maria da Penha* é instância de *Lei* e *João* é instância de *Pessoa*. Neste relacionamento, a categoria classifica a entidade.

O relacionamento de especialização, em contrapartida, indica uma relação de hierarquia entre entidades. Por exemplo, *Pessoa* especializa *Indivíduo*. As categorias do nível conceitual são instâncias do nível meta-conceitual. Os indivíduos do domínio são instâncias das categorias do domínio. A especialização ocorre entre indivíduos de mesmo nível.

O relacionamento de instanciação propaga-se por meio do relacionamento de especialização. Deste modo, no exemplo apresentado na Figura 2.7, *Rex* é instância de *Cachorro*, mas também instância de *Indivíduo*. Do mesmo modo, *Cachorro* é instância de *Universal Monádico* assim como *Indivíduo* também é instância de *Universal Monádico*.

2.6.2 Substanciais e Momentos

A categoria *Universal Monádico* é especializada por outras duas: *Momento Universal (Moment Universal)* e *Universal Substancial (Substantial Universal)*. Instâncias de *Universal Substancial* formam a categoria de particulares chamada de *Substanciais* (*Substantial)*[11]. Instâncias de *Momento Universal* formam a categoria de particulares *Momento (Moment)*[12].

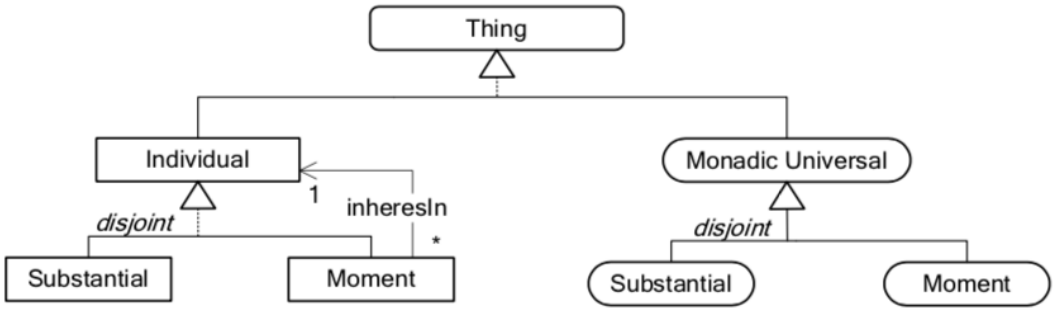


Figura 2.8: Fragmento da UFO exibindo universais e indivíduos dos tipos *Substanciais* e *momentos*. Fonte: Zamborlini [67]

Os particulares da categoria *Substancial* são indivíduos existencialmente independentes. Os particulares da categoria *Momento*, por outro lado, são indivíduos existencialmente dependentes e inerentes a outro indivíduo – em que este pode ser um *momento* ou um *substancial*. A Figura 2.8 apresenta o relacionamento de inerência (*inheresIn*) entre

[11]Em alguns artigos, utiliza-se *Objeto (Object)* para se referir a *Substanciais* [95].
[12]Também chamados de *trope* [30].

momentos e *indivíduos*. *Momentos* podem ser existencialmente dependentes de outros momentos, formando uma cadeia de indivíduos. Entretanto, o polo mais geral dessa cadeia deve necessariamente ser um indivíduo independente existencialmente, ou seja, um *substancial*. São *momentos*: a dor de cabeça de João (*momento* inerente ao *substancial* João) e a intensidade da dor de cabeça de João (*momento* inerente à dor de cabeça de João). A Figura 2.9 apresenta exemplos de *substanciais* e *momentos*.

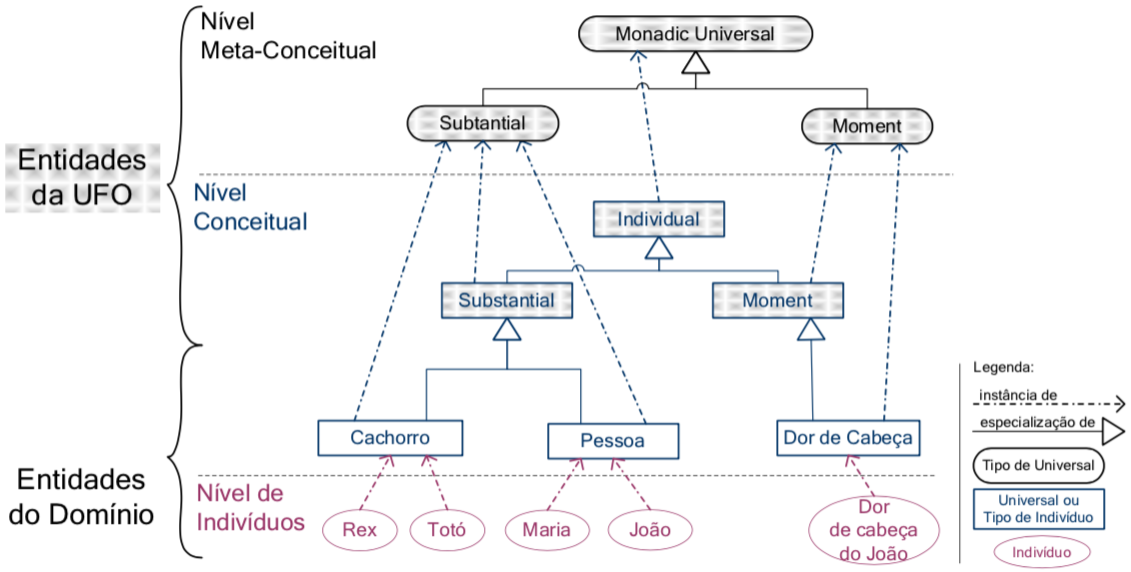


Figura 2.9: Análise em níveis dos tipos de universais e indivíduos: *Substanciais* e *momentos*. Fonte: Zamborlini [67]

2.6.3 Subcategorias de Substanciais Universais

A Figura 2.10 apresenta as categorias que especializam o tipo de universal *Substancial Universal*. A categoria *Substancial Universal* é dividida em duas categorias: *Sortal Universal* e *Mixin Universal*, de acordo com as noções de identidade e rigidez apresentados na Seção 2.5.

Sortais agregam indivíduos com o mesmo princípio de identidade [32, 67]. Por exemplo, *Pessoa*, *Cidadão*, *Pessoa Física* possuem o mesmo princípio de identidade: elas podem ser identificadas pela digital do indivíduo. Portanto, *Pessoa*, *Cidadão*, *Idoso* são *sortais*.

Mixins, por sua vez, agregam indivíduos com princípios de identidade diferentes. Um *Item catalogável*, por exemplo, pode ser um *Compact Disc* – identificado pelo *CDDB* – ou um *Livro* – identificado pelo *ISBN*. Logo, neste cenário, *Item catalogável* é um *mixin*.

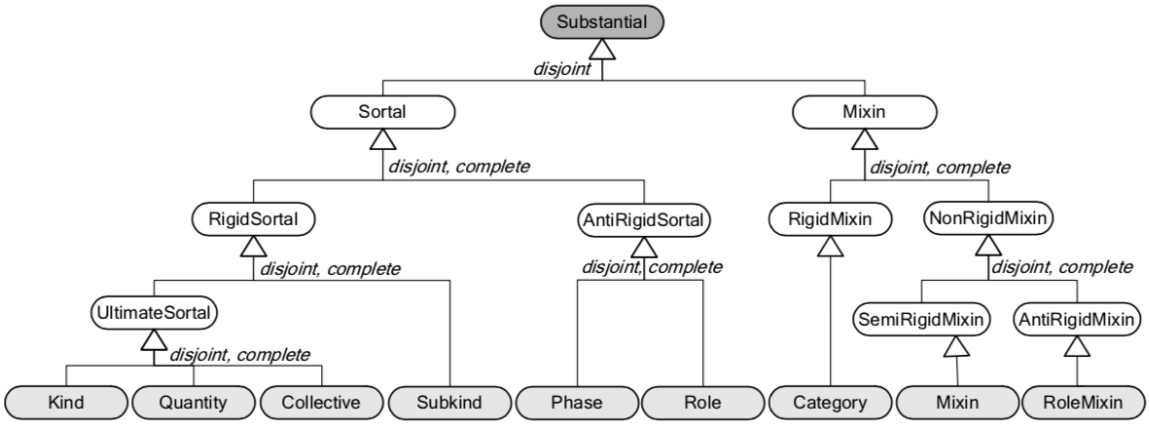


Figura 2.10: Especialização do tipo de universais *Substancial*. Fonte: Zamborlini [67]

2.6.4 Sortais

A UFO especializa a categoria de Universais chamada sortal em dois grupos: *sortais rígidos* (*Rigid Sortal*) e *sortais anti-rígidos* (*Anti-rigid Sortal*). Instâncias de sortais rígidos sempre serão sortais rígidos. São exemplos de sortais rígidos: *árvore*, *animal*, *automóvel*. Instâncias de árvore sempre são árvores. Instâncias de animal não deixam de ser um animal. O mesmo vale para instâncias de automóvel.

Em contrapartida, instâncias de *sortais anti-rígidos* podem deixar de instanciá-los eventualmente. Exemplos de *sortais anti-rígidos* são: *empregado*, *aluno*, *cliente*, *adolescente*, *vivo*, etc. Um *aluno* pode deixar de ser *aluno* caso seu vínculo com uma *instituição de ensino* termine. Um *adolescente* deixa de ser *adolescente* quando se torna *adulto*. Um *indivíduo vivo* deixa de ser vivo quando morre.

Os sortais rígidos são classificados em dois grupos: *Ultimate Sortal* e *Subespécie (Subkind)*. O tipo *Ultimate Sortal* possui o princípio de identidade utilizado por suas instâncias. *Subespécies*, por outro lado, apenas herdam esse princípio. Caso a placa de identificação de um veículo seja o princípio de identidade de veículo, então *veículo* é um *ultimate sortal*. *motocicleta*, *carro* e *caminhão* são *subespécies*.

Há três subcategorias de *ultimate sortals* descritos na UFO: *Espécie (Kind)*, *Quantidade (Quantity)* e *Coletivo (Collective)*. *Espécies* são *ultimate sortals* que fornecem um princípio de identidade para suas instâncias [32, p. 108]. O universal *Quantidade* representa *ultimate sortals* cujas instâncias são porções maximais de uma quantidade de matéria. *Quantidades* consistem de universais cujas instâncias são tipicamente referenciadas em linguagem natural utilizando o conceito de massa (prata, diamante etc) [67, 99]. O universal *Coletivo*, por sua vez, é aquele cujas instâncias constituem-se de uma quantidade de entidades que juntas formam um todo com uma estrutura uniforme [32, 100].

Quanto aos *sortais anti-rígidos*, estes são especializados em duas categorias: *Fase (Phase)* e *Papel (Role)* (Figura 2.10). *Fases* são possíveis etapas na história de um particular [95]. Alguns exemplos de *Fases*:

- *Vivo* e *Morto* são fases de *Ser Vivo*;
- *Ovo*, *Larva*, *Pupa* e *Adulto* são fases de *Abelha*.

Para Guizzardi e Wagner [95, tradução nossa], "Papéis diferem de fases em relação à condição de especialização φ. [...] Para uma fase $\mathcal{F}$, φ representa uma condição que depende unicamente das propriedades intrínsecas de $\mathcal{F}$. Para um papel $\mathcal{P}$, por outro lado, φ depende das propriedades extrínsecas (relacionais) de $\mathcal{P}$. Por exemplo, pode-se dizer que, se João é um aluno, João é uma pessoa que está matriculada em uma instituição de ensino, se Pedro é um cliente, Pedro é uma pessoa que compra um produto x de um fornecedor y, ou se Maria é uma Paciente, então ela é uma pessoa tratada em uma determinada unidade médica[13]."

2.6.5 *Mixins*

Os universais do tipo *Mixin* dividem-se em dois grupos: *Mixin Rígido* (*Rigid Mixin*) e *Mixin Não-rígido* (*Non-Rigid Mixin*), conforme o critério de rigidez apresentado na Seção 2.5.

Mixins Rígidos são também conhecidos como *Categorias*. Eles representam abstrações de propriedades rígidas comuns a dois ou mais sortais [30]. *Item catalogável* é um *Mixin rígido*, pois indivíduos com princípios de identidade distintos, como *Compact Discs* e *Livros*, podem ser itens catalogáveis.

Por outro lado, *Mixins* não-rígidos são divididos em dois grupos: *Mixin Semi-rígido* (*Semi-Rigid Mixin*) e *Mixin Anti-rígido* (*Anti-Rigid Mixin*). Os *Mixins Semi-rígidos* são também chamados apenas de *Mixins* enquanto os *Mixins Anti-rígidos* são chamados de *Mixin de papel (Role Mixins)* [67].

Mixins semi-rígidos representam propriedades que são essenciais para algumas entidades, mas acidentais para outras [32, p. 113]. Guizzardi cita como exemplo desta categoria o conceito *sentável*, o qual é essencial para *cadeira*, mas acidental para *caixote*. Por fim, *Mixins anti-rígidos* "representam abstrações de propriedades comuns de dois ou mais

[13] *Roles differ from phases with respect to the specialization condition φ. For a phase Ph, φ represents a condition that depends solely on intrinsic properties of Ph. [...] For a role Rl, conversely, φ depends on extrinsic (relational) properties of Rl. For example, one might say that if John is a Student then John is a Person who is enrolled in some educational institution, if Peter is a Customer then Peter is a Person who buys a Product x from a Supplier y, or if Mary is a Patient than she is a Person who is treated in a certain medical unit.*

sortais anti-rígidos (*Anti Rigid Sortal*) que especializam ao menos duas Espécies (*Kind*) distintas" [30, p. 437].

2.6.6 Subcategorias de *momentos*

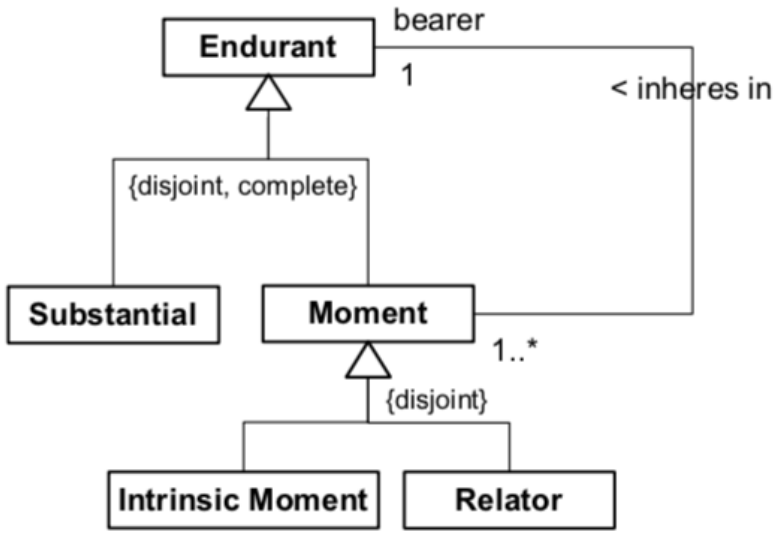


Figura 2.11: Relação de *momentos* com seus portadores. Fonte: Guizzardi [32, p. 215]

Guizzardi [32, p. 212] afirma que a noção de *momentos* foi descrita pela primeira vez na teoria de acidentes individuais nas obras *Metafísica* e *Categorias*, ambas de Aristóteles. Para o filósofo grego, um acidente é uma propriedade individualizada, evento ou processo que não é parte de uma coisa. Na tese de Guizzardi, o termo *momento* é usado em um sentido um pouco mais geral e abrange [32, p. 213, tradução nossa]:

1. *Momentos Intrínsecos* (ou *Intrinsic Moments*): *qualidades (qualities)* como uma cor, um peso, uma carga elétrica, uma forma circular; *modes* como um pensamento, uma habilidade, uma crença, uma intenção, uma dor de cabeça, assim como disposições tais como a propriedade de refrangibilidade dos raios luminosos, ou a disposição de um material elétrico de atrair objetos metálicos[14];
2. *Momentos Relacionais (ou Relators)*: um beijo, um aperto de mão, uma ligação covalente, mas também objetos sociais tais como uma conexão de vôo, um pedido de compra e um compromisso ou reivindicação[15].

[14]Intrinsic Moments*: qualities such as a color, a weight, a electric charge, a circular shape; modes such as a thought, a skill, a belief, an intention, a headache, as well as dispositions such as the refrangibility property of light rays, or the disposition of a magnetic material to attract a metallic object.*

[15]Relational Moments (or relators)*: a kiss, a handshake, a covalent bond, but also social objects such as a flight connection, a purchase order and a commitment or claim*

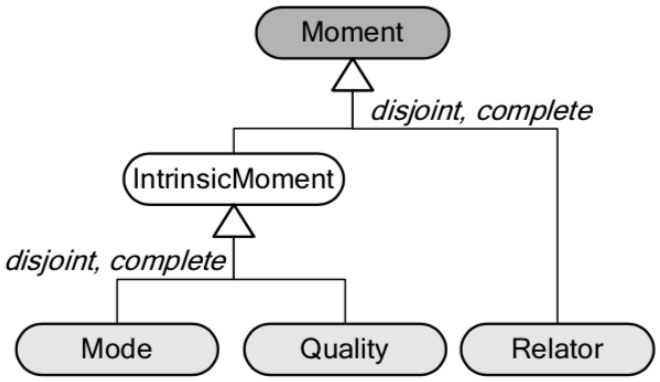


Figura 2.12: Fragmento da UFO exibindo a especialização do tipo de universais *momentos*. Fonte: Zamborlini [67]

Um *momento* é dependente existencialmente de indivíduo. Esta dependência é caracterizada por uma relação de *inerência* entre o *momento* e o *indivíduo*. Deste modo, distingue-se *momentos intrínsecos* e *relacionais* da seguinte forma:

- *momentos intrínsecos* são existencialmente dependentes de um único indivíduo;
- *momentos relacionais* dependem existencialmente de múltiplos indivíduos.

Os *Momentos Intrínsecos (Intrisic Moments)* são especializados em mais duas categorias de acordo com a existência ou não da relação entre o *momento* e uma *dimensão de qualidade (quality dimension)*. Alguns *Momentos Intrínsecos* estão associados com estruturas com uma ou mais dimensões na cognição humana. Por exemplo, massa, idade, circunferência, volume são *momentos intrínsecos* associados a uma estrutura unidimensional com valores possíveis no conjunto de números reais positivos. Outros *momentos intrínsecos* estão associados a estruturas multidimensionais como cor, paladar e cheiro. Ambas as estruturas são chamadas de *domínios de qualidade (quality domains)* e compreendem uma ou mais dimensões de qualidade.

Assim, *qualidades* são *momentos intrínsecos* associados a um *domínio de qualidade*. Por sua vez, *Modos (Modes)* são *momentos intrínsecos* que não estão diretamente relacionados com *estruturas de qualidade (quality structures)*[16] [32, p. 237, tradução nossa]. Exemplos de *qualities*: cor, idade, altura. Exemplos de *modos*: crenças, fé, febre. A Figura 2.13 apresenta um exemplo de Zamborlini [67] com os indivíduos, os tipos de indivíduos e os tipos de universais derivados de *Momento*.

Um tipo específico de modo extrínseco é chamado de *qua indivíduo*. De acordo com Guizzardi [32, p.238 - 240], um *qua indivíduo z* é portador de todos os *modos* (*modes*)

[16] *"Modes are intrinsic moments that are not directly related to quality structures"*

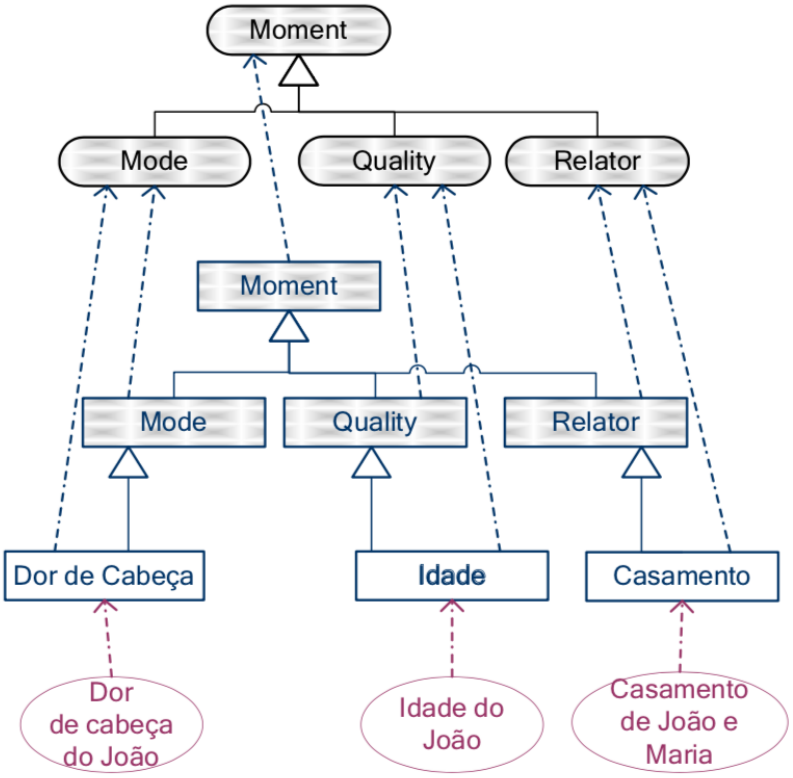


Figura 2.13: Esquema de níveis - *momentos*. Fonte: Zamborlini [67]

externamente dependentes de outro indivíduo y portador de z, desde que esses modos compartilhem as mesmas dependências e a mesma base. O autor explica que é possível agregar todos os qua indivíduos que compartilham a mesma base em um único *relator*.

Assim, Guizzardi [32, p. 240, tradução nossa] afirma:

> Dados três indivíduos distintos x, y e z de modo que: (a) x é um relator; (b) y é um qua indivíduo e y é parte de x; (c) y é inerente a z. Neste caso, podemos dizer que x media z, simbolizado por m. Formalmente, temos que[17]:

$$\forall x,y\ m(x,y) \implies relator(x) \wedge Endurant(y) \tag{2.1}$$

$$\forall x\ Relator(x) \implies \forall y\ (m(x,y) \iff (\exists z\ quaIndividual(z) \wedge (z < x) \wedge i(z,y))) \tag{2.2}$$

Da Equação 2.2, tem-se que para todo *Relator* x implica que para todo y há uma relação de *mediação* entre x e y se, e somente se, existir um *qua indivíduo* z de tal modo que z é parte de x e z é inerente a y.

[17] let x, y and z be three distinct individuals such that: (a) x is a relator; (b) y is a qua individual and y is part of x; (c) y inheres in z. In this case, we say that x mediates z, symbolized by m. Formally, we have that:

Portanto, *Relators* são tipos particulares de momentos compostos de modos externamente dependentes chamados de *qua indivíduos*, enquanto "qua indivíduos são modos (potencialmente complexos) dependentes externamente que exemplificam todas as propriedades que um indivíduo possui no escopo de certa relação material[18]" [32, p. 242].

2.6.7 Universais de Relação e Categoria de Instâncias de Relações

A tradição filosófica costuma dividir as relações em dois grupos: *relações formais* e *relações materiais*. As relações formais existem por meio da conexão direta entre os indivíduos relacionados. As relações materiais dependem de uma entidade mediadora [57]. Por exemplo, uma relação *casado com* necessita de uma entidade reificada[19] chamada *casamento* que carrega uma série de propriedades da própria entidade *casamento*. Por isso, *casado com* é uma relação material. Já a relação *Carlos é mais velho que Pedro* é formal, pois não depende de um intermediador para a relação se concretizar.

Conforme apresentado na Figura 2.14, a UFO segue a tradição filosófica. O tipo *Relation Universal* divide-se nas mesmas duas categorias: *Relação Material (Material Relation)* e *Relação Formal (Formal Relation)*.

Nesta ontologia de fundamentação, a entidade intermediadora de relações materiais é chamada de *Relator* [32, p. 236-237]. Por exemplo, a relação material *Estuda em* entre *Maria* e *Universidade de Brasília* só existe enquanto também o *relator Matrícula de Maria na Universidade de Brasília* estiver ativa. *Relators* foram discutidos na Subseção 2.6.6.

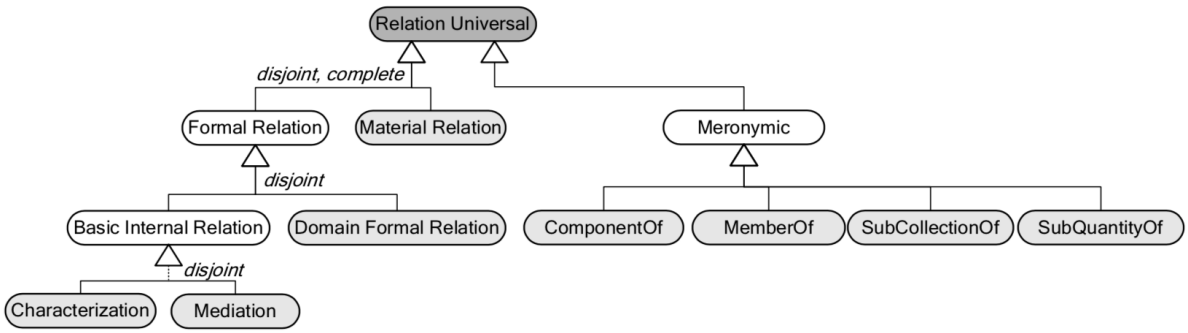


Figura 2.14: Fragmento da UFO: tipos de Relações. Fonte: Zamborlini [67]

Quanto às *Relações Formais*, Zamborlini [67, p. 40, grifos da autora] afirma:

[18] *qua individuals are (potentially complex) externally dependent modes exemplifying all the properties that an individual has in the scope of a certain material relation.*

[19] De acordo com Houaiss et al. [101, p. 1636], reificar é "encarar (algo abstrato) como uma coisa material ou concreta; coisificar"

As relações do tipo **Formal** podem ser ainda classificadas como **Relação Básica Interna** (*Basic Internal Relation*) ou **Relação Formal de Domínio** (*Domain Formal Relation*). O tipo *Basic Internal Relation* aplica-se a relações formais internas ditas de dependência existencial que têm representação entre as categorias de Indivíduos da UFO. Este tipo se subdivide em **Caracterização** (Characterization), que se aplica à relação de inerência (*inheresIn*) que define os indivíduos do tipo *Moment*, e **Mediação** (*Mediation*), que se aplica à relação de mediação (mediates) que define os indivíduos do tipo Relator. Por sua vez, o tipo *Domain Formal Relation* se aplica às relações formais que são específicas de domínio e que, por isso, não são representadas entre os tipos de indivíduos da UFO, mas entre os conceitos específicos de domínio, assim como as do tipo *Material Relation*.

Transversalmente à classificação das relações em formal e material, as relações são classificadas por meio de critérios meronímicos (relações parte/todo). Aspectos meronímicos da UFO são estudados em Guizzardi [53, 99, 102].

2.7 UFO-B: diferença entre indivíduos endurantes e indivíduos perdurantes

Mais uma vez, é importante destacar que a UFO-B (Figura 2.15) é uma ontologia de perdurantes. A UFO-A, conforme mencionado na Seção 2.6, é uma ontologia de endurantes. A UFO-B é tratada de forma completa em Guizzardi et al. [91] e, mais recentemente, em [103].

Johansson [105] distingue indivíduos endurantes e indivíduos perdurantes da seguinte maneira:

1. Indivíduos endurantes:
   (a) persistem;
   (b) não possuem partes temporais;
   (c) estão necessariamente presentes em cada instante de tempo no qual existem.
2. Indivíduos perdurantes, por sua vez:
   (a) persistem;
   (b) possuem partes temporais;
   (c) necessariamente não estão presentes de modo completo em cada instante de tempo no qual existem.

Enquanto um indivíduo endurante continuar a ser (persistir), *ele mantém sua identidade imutável, mesmo que suas propriedades mudem.* Por exemplo, um *carro c* continua a

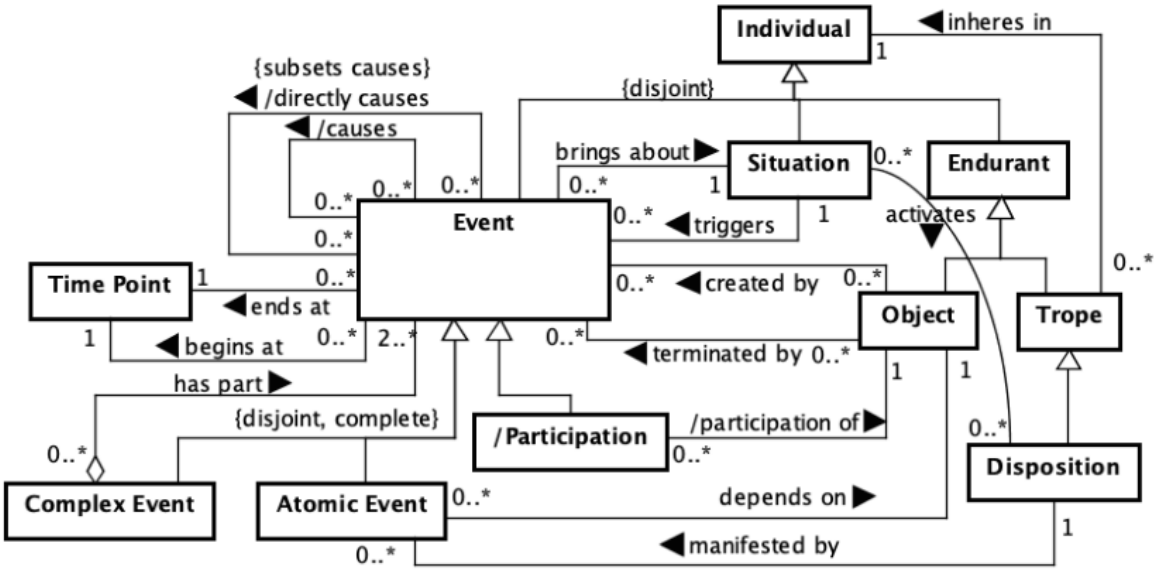


Figura 2.15: Um resumo das categorias da UFO-B. Fonte: Almeida et al. [104]

ser o mesmo carro mesmo que suas rodas aro 15 sejam trocadas por rodas aro 16. Apesar da alteração em cada propriedade *roda do carro c*, a identidade do *carro c* continua a mesma. Como um tipo endurante não pode ter partes temporais, ele necessariamente está presente em qualquer tempo em que continue a ser (em que persista).

Já perdurantes (eventos e processos) também persistem. Entretanto são imutáveis e possuem, necessariamente, partes temporais. Consequentemente, não estão completamente presentes em qualquer instante de tempo em que continuem a ser.

A UFO-B possui cinco subteorias [91, 104]:

1. mereologia: relações parte/todo;
2. participação de objetos em eventos: eventos são externamente dependentes em endurantes;
3. relações temporais: eventos ocorrem no tempo e possuem propriedades temporais;
4. mapeamento de situações para situações na realidade: representação de mudanças na realidade;
5. manifestações de disposições de objetos: trata da conexão entre endurantes e perdurantes por meio das manifestações de certas disposições de objetos.

Mereologia de Eventos

Perdurantes podem conter outros perdurantes. Por exemplo, o evento *Copa do Mundo FIFA de 2018* é composto de vários outros eventos tais como: *Cerimônia de Abertura*, *partida de Croácia contra Argentina*, *Gol de Cavani na partida de Uruguai contra Rússia*, *a consulta ao Árbitro Assistente de Vídeo (Video Assistant Referees (VAR)) para validar o gol de Diego Costa na partida entre Espanha e Portugal*, *Substituição de Ante Rebić por Andrej Kramarić na partida final da copa* etc.

A Figura 2.16 mostra que, dependendo de sua estrutura mereológica, um evento pode ser atômico – não possuem outros eventos – ou complexo – são agregações de pelo menos dois eventos disjuntos [91].

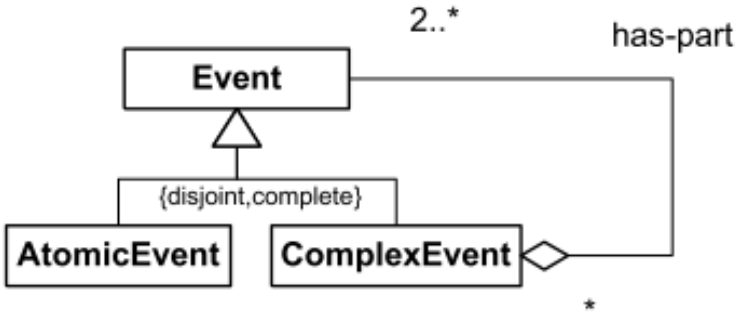


Figura 2.16: Eventos Atômicos e Complexos. Fonte: Guizzardi et al. [91]

Participação de Objetos em Eventos

Guizzardi et al. [91, tradução nossa] argumentam que "eventos são entidades ontologicamente dependentes no sentido de que são existencialmente dependentes de objetos. Como discutido anteriormente, os eventos podem ser (mereologicamente) atômicos ou complexos. Diz-se que um evento atômico é diretamente existencialmente dependente de um objeto. Essa relação (denominada aqui *depende de*) é a contraparte perdurante da relação de inerência entre tropes e seus portadores[20]."

Ainda de acordo com Guizzardi et al. [91], eventos complexos são existencialmente dependentes tanto dos eventos que o formam quanto indiretamente dependentes dos objetos de que suas partes dependem. Isso permite aos autores apresentarem uma forma ortogonal de particionamento de eventos: aquela em que um evento pode ser dividido em

[20] *Events are ontologically dependent entities in the sense that they existentially depend on objects in order to exist. As previously discussed, events can be either (mereologically) atomic or complex. An atomic event is said to be directly existentially dependent on an object. This relation (termed here dependsOn) is the perdurant counterpart of the inherence relation between tropes and their bearers.*

partes que são existencialmente dependentes de cada um dos seus participantes, conforme aponta a Figura 2.17.

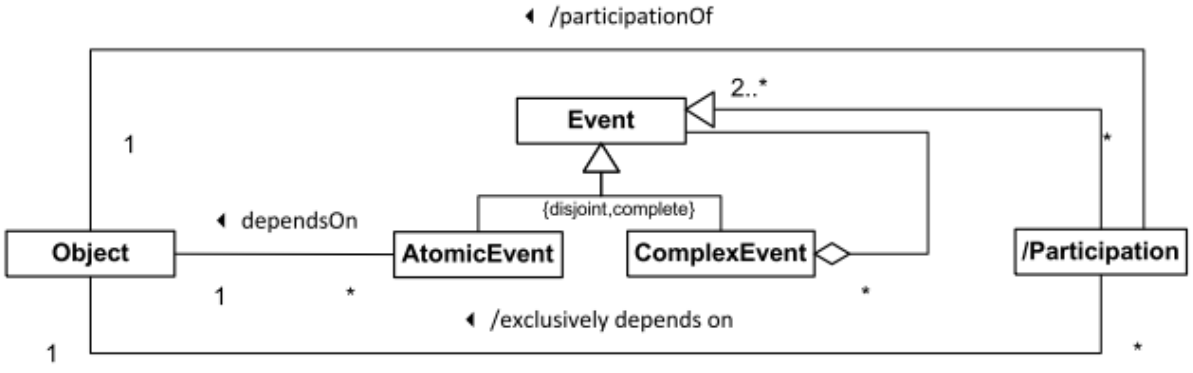


Figura 2.17: Eventos complexos como a soma de participação de objetos. Fonte: Guizzardi et al. [91]

Guizzardi et al. [91] chamam de *participação* a parte de um evento que depende exclusivamente de um único objeto. Como esta classificação é independente da anterior, uma participação pode, também, ser atômica ou complexa.

Relações temporais entre eventos

Toda propriedade temporal de objetos está relacionada a eventos nos quais estes objetos participam. Eventos possuem algumas relações com o espaço temporal. Cada evento possui um início e um fim, os quais podem ou não ser coincidentes. O início e o fim de um evento apontam para algum espaço temporal. Eventos que contenham outros eventos devem, necessariamente, não começar depois nem terminar antes de seus subeventos [91].

Representação de mudanças

Eventos transformam uma porção da realidade em outra porção da realidade [91]. Para explicar mudanças na realidade, Guizzardi et al. [91, tradução nossa] utilizam-se da noção de situação: "uma situação é uma configuração particular de uma parte da realidade que pode ser entendido como um todo.[21]". As situações estão indexadas num espaço temporal e podem ser factuais – caso em que são chamadas de *fatos* – ou contrafactuais. Situações contrafactuais estão fora do escopo deste texto. A Figura 2.18 exibe o relacionamento entre uma situação e um evento.

Guizzardi et al. [91, tradução nossa] postulam duas relações possíveis entre situações e eventos: "(i) uma situação s desencadeia um evento e, caso e ocorra devido à obtenção

[21] A situation is a particular configuration of a part of reality which can be understood as a whole.

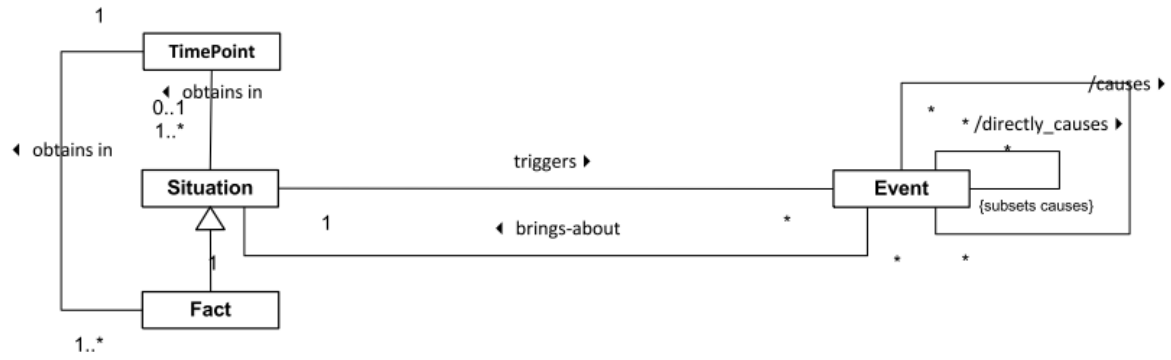


Figura 2.18: Situações e seus relacionamentos com pontos no tempo. Fonte: Guizzardi et al. [91]

de s; (ii) um evento provoca uma situação s. Nesse caso, a ocorrência de um evento e resulta na obtenção da situação s no mundo em um ponto temporal específico ao fim do evento, ou seja, resulta em s tornando-se um fato em um ponto específico do tempo que coincide com o do fim do evento e. Uma relação de desencadeamento entre a situação s e o evento e captura a noção que s exemplifica um estado do mundo que satisfaz todas as condições suficientes e necessárias para a manifestação de e.[22]"

Manifestações de disposições

Sobre a manifestação de disposições, Guizzardi et al. [91, tradução nossa] se manifestam da seguinte maneira:

> Seguindo Mumford [106], nós consideramos disposições como propriedades que apenas se manifestam em situações particulares e que também podem deixar de se manifestar. Quando manifestadas, elas se manifestam por meio da ocorrência de eventos. Tomemos, por exemplo, a disposição de um ímã m para atrair material metálico. O objeto m possui essa disposição mesmo que nunca se manifeste, por exemplo, porque nunca está próximo de nenhum material magnético. Não obstante, pode-se dizer que certamente m possui aquela propriedade intrínseca (até mesmo essencial, neste caso) que compartilha com outros ímãs. Agora, um material metálico particular também tem a disposição de ser atraído por ímãs. Dada uma situação em que m está na presença de um objeto metálico em particular (a uma certa distância, de uma certa massa, em uma superfície com um certo atrito etc.), as disposições dessas duas entidades (objeto metálico, ímã) podem se manifestar

[22] *(i) a situation s triggers an e event, in the case that e occurs because of the obtaining of s, and; (ii) an event brings about a situation s, in which case the occurrence of an event e results in the situation s obtaining in the world at the time point end-point(e), i.e., results in s becoming a fact in end-point(e). A triggers relation between situation s and event e captures the notion that s exemplifies a state of the world that satisfies all the sufficient and necessary conditions for the manifestation of e.*

por meio da ocorrência de um evento complexo, a saber, o movimento desse objeto em direção ao ímã. [23].

A Figura 2.19 sumariza a subteoria de eventos atômicos como manifestações de disposições de objetos.

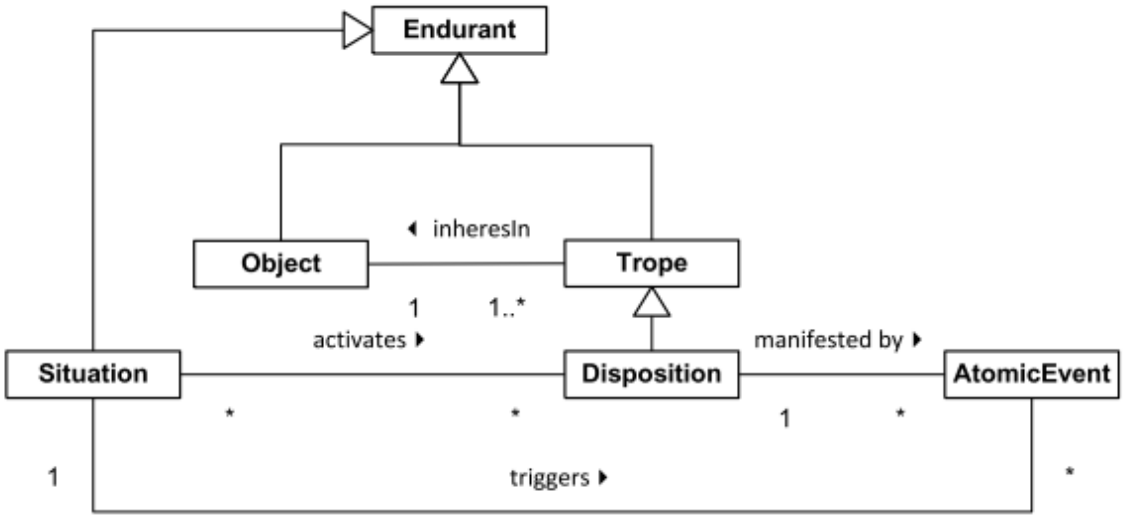


Figura 2.19: Eventos atômicos como manifestações de disposições de objetos. Fonte: Guizzardi et al. [91]

2.8 Restrições para categorias UFO

A seguir, apresentam-se algumas restrições para verificar se o modelo de objetos produzido atende aos axiomas das categorias da UFO utilizadas neste trabalho. As restrições aqui descritas são adaptações daquelas presentes em Guizzardi [32, p. 317 - 352] e Araujo [30, p. 196 - 226], porém no contexto de banco de dados relacional, visto que nas obras originais as restrições aplicam-se à OntoUML[24].

[23] *Following [106], we consider dispositions as properties that are only manifested in particular situations and that can also fail to be manifested. When manifested, they are manifested through the occurrence of events. Take for example the disposition of a magnet m to attract metallic material. The object m has this disposition even if it is never manifested, for example, because it is never close to any magnetic material. Nonetheless, m can certainly be said to possess that intrinsic (even essential, in this case) property, which it shares with other magnets. Now, a particular metallic material has also the disposition of being attracted by magnets. Given a situation in which m is in the presence of a particular metallic object (at a certain distance, of a certain mass, in a surface with a certain friction, etc.), the dispositions of these two entities (metallic object, magnet) can be manifested through the occurrence of a complex event, namely, the movement of that object towards the magnet.*

[24] https://ontouml.org

2.8.1 Substance Sortal

Substance Sortal é uma meta categoria abstrata que representa propriedades gerais – como rigidez (*rigidity*) – de todos os *substanceSortal* relacionalmente independentes de objetos universais que forneçam um princípio de identidade para suas instâncias. A categoria *Substance Sortal* é um sortal rígido que não possui sintaxe concreta. Dessa forma, não devem existir instâncias diretas de um *Substance Sortal*.

Tipo superior de

Kind (Subseção 2.8.2);

Quantity (não utilizada neste texto);

Collective (não utilizada neste texto).

Restrições

1. Todo *objeto substancial* deve ser uma instância de um *substanceSortal*, direta ou indiretamente. Isso significa que todo elemento concreto do banco de dados deve estar numa hierarquia que inclui uma categoria de domínio que seja um *kind*, um *quantity* ou um *collective*;
2. Um *objeto substancial* não deve instanciar mais do que um *substanceSortal*;
3. Um *substanceSortal* não pode incluir outro *substanceSortal* nem um *subkind* (Subseção 2.8.3) em sua hierarquia de tipos, ou seja, um *substanceSortal* não pode ter uma supertipo membro de $\{ kind, subkind, quantity, collective \}$;
4. Um tipo que represente um *universal substancial rígido* (*rigidSortal*) não pode ser um subtipo de um tipo que representa um *universal anti-rígido*. Dessa forma, um *substanceSortal* não pode ter como supertipo um membro de $\{ phase, role, roleMixin \}$.

2.8.2 Kind

Um *Kind* representa um *substanceSortal* (Subseção 2.8.1) que provê princípio de identidade a suas instâncias [32, p. 108].

Exemplos

Tipos Naturais (como Pessoa, Gato, Flor) e artefatos (Cadeira, Carro, Celular).

Subtipo de

substanceSortal (Subseção 2.8.1)

Restrições

(todas as restrições do Subseção 2.8.1 se aplicam)

2.8.3 Subkind

Um *subkind* é uma restrição relacionalmente independente de um *substanceSortal* (Subseção 2.8.1) que carrega um princípio de identidade fornecido por este.

Exemplos

Um exemplo de *subkind* pode ser `Homem` e `Mulher` como tipos de um *kind* `Pessoa`.

Subtipo de

Sortal Rígido

Restrições

1. Como um tipo que representa um universal rígido não pode especializar (restringir) um tipo anti-rígido [32, p. 103], um *subkind* não pode ter como tipo superior um membro de $\{ phase, role, roleMixin \}$;
2. Todo *subkind* deve estender um *kind*[a].

[a] Esta regra era implícita pela própria definição de *subkind*. Nota de Araujo [30]

2.8.4 Abstract Mixin Type

A *Abstract Mixin Type* (*mixinUniversal*) é uma metacategoria abstrata que representa propriedades gerais de todos os *mixins*. *Mixin class* não possuem sintaxe concreta e, portanto, não possuem instâncias diretas em banco de dados.

Tipo superior de

category (Subseção 2.8.5);

roleMixin (categoria não utilizada neste texto);

mixin (Subseção 2.8.6);

Restrições

1. Tipos que representam *nonSortal* não podem ser subtipos de um tipo que representa um *sortal*. Como consequência, um *abstract mixin type* não pode ter como tipo superior um membro de { *kind*, *quantity*, *collective*, *subkind*, *phase*, *role* };
2. Qualquer tipo que instanciar algum dos *nonSortal* não podem ter instâncias diretas. Dessa forma, qualquer instância de um dos *mixin class* não pode ter instâncias diretas.

2.8.5 Category

Um *category* representa um *Mixin rígido* relacionalmente dependente, ou seja, trata-se de um *dispersiveSortal* que agrega propriedades essenciais que são comuns a diferentes *substanceSortal* (Subseção 2.8.1).

Exemplos

A categoria `SerVivo` como uma generalização de `SerHumano` e `Palmeira`.

Subtipo de

Mixin class (Subseção 2.8.4)

Restrições

1. Pelo fato de representar um universal rígido (que não pode especializar ou restringir um tipo anti-rígido[a]), e também pela combinação das regras do Subseção 2.8.4, um *category* não pode ter como tipo superior um membro de { *kind*, *quantity*, *collective*, *subkind*, *phase*, *role*, *roleMixin* };
2. Um *category* nunca possui instâncias diretas;

3. Um *category* só pode ser subdividido em outro *category* ou em um *mixin*.

[a][32, p. 103]

2.8.6 Mixin

Um *mixin* representa um conjunto de propriedades que são essenciais para algumas de suas instâncias e acidentais para outras.

Exemplos

Um exemplo é o *mixin* `Sentável`, que representa uma propriedade que pode ser considerada essencial para *kinds* (Subseção 2.8.2) como `Cadeira` e `Banco`, mas são acidentais para `Engradado`, `CaixaDePapel` ou `Pedra`.

Subtipo de

Mixin class (Subseção 2.8.4)

Restrições

1. Um *mixin* não pode ter um *roleMixin* como um tipo superior;
2. Um *mixin* nunca possui instâncias diretas;

2.8.7 Moment Type

Trata-se de uma meta categoria abstrata que representa propriedades gerais de todos os *momentUniversal*. *moment* não possui sintaxe concreta. Dessa forma, não possuem instâncias diretas em banco de dados.

Tipo superior de

intrinsicMomentUniversal

quality (Subseção 2.8.11);

mode (Subseção 2.8.9);

relator (Subseção 2.8.10).

Restrições

1. Todo *moment* deve (direta ou indiretamente) estar conectado a uma associação com pelo menos uma relação do tipo *characterization*.

2.8.8 Quale

O *quale* é representação de um ponto no *espaço conceitual* representado pelo *qualityDomain* ou *qualityDimension*. Trata-se de um conceito que diz respeito apenas a *particular*.

Restrições

1. Um *quale* é relacionado via *attributeFunction* (Subseção 2.8.11) a uma instância de um *endurant* e a um *valor abstrato* contido no *espaço conceitual* do *qualityStructure* co-domínio da *attributeFunction*;
2. O valor do *quale* deve atender às restrições impostas pelo *qualityStructure* ao qual se refere.

2.8.9 Mode Type

Um *mode* é instância de um *modeUniversal*, um tipo de *intrinsicMomentUniversal*. Toda instância de um *modeUniversal* é existencialmente dependente (*existentialDependence*) de exatamente uma entidade.

Ver também Subseção 2.8.11.

Exemplos

Habilidades, pensamentos, crenças, intenções, sintomas.

Subtipo de

momentUniversal (Subseção 2.8.7)

intrinsicMomentUniversal

Restrições

1. Por ser um *moment*, deve (direta ou indiretamente) estar conectado a uma associação com pelo menos uma relação do tipo *characterization*;

2.8.10 Relator Type

Um *relator type* (*relatorUniversal*) é um *momentUniversal*, cujas quaisquer instâncias são existencialmente dependentes de ao menos duas entidades. *relator* são as instâncias de propriedades relacionais.

Exemplos

Casamento, beijo, aperto de mão, acordos, compras.

Subtipo de

momentUniversal (Subseção 2.8.7)

Restrições

1. Todo *relator* deve (direta ou indiretamente) estar conectado a uma associação com pelo menos uma relação *mediation*;
2. Todo relator deve mediar ao menos duas entidades diferentes, dessa forma, seja R uma categoria instância de *relator*; $\{C_1, \ldots, C_n\}$ um conjunto de *universais* relacionados a R via uma relação *mediation* (*universal* mediados por R); $lower_{C_i}$ sendo $(1 \leq i \leq n)$ o valor mínimo da restrição de cardinalidade da associação conectada à C_i na relação *mediation*. Dessa forma, tem-se a restrição:
$$(\sum_{i=1}^{n} lower_{c_i}) \geq 2$$
3. Além da restrição anterior, adicionalmente, ao menos dois *universais* mediados por R relacionados ao *relator* devem ser diferentes;
4. Todo *moment* deve (direta ou indiretamente) estar conectado a uma associação com pelo menos uma relação do tipo *characterization*;

2.8.11 Quality Type

Um *quality type* (*qualityUniversal*) representa um *intrinsicMoment* relacionado a um *qualityStructure* e é existencialmente dependente de exatamente uma entidade.

Ver também Subseção 2.8.9.

Subtipo de

momentUniversal (Subseção 2.8.7)

intrinsicMomentUniversal

Restrições

1. Todo tipo *quality* deve estar associado a pelo menos um *qualityStructure*;
2. Toda instância de *quality* deve estar associado a um *endurant*, no sentido que o *qualityParticular* é inerente ao *endurant*, o que é simétrico ao *endurant* possuir o *qualityParticular*;
3. Os *quality* podem ser representados via *attributeFunction*[a];
4. Uma *attributeFunction* deve ser representada como propriedade do *classifier* que a possui, no caso de o co-domínio da função ser um *qualityDimension*;
5. Uma *attributeFunction* deve ser representada como relação direcionada partindo do *classifier* ao co-domínio, no caso de o co-domínio da função ser um *qualityDomain*
6. *quality* instanciam um único *qualityUniversal*;

[a] [32, p. 327].

2.9 Multi-level Theory

Normalmente, modelos conceituais possuem dois níveis de indivíduos: classes e instâncias. Entretanto, em alguns casos, há modelos com níveis de categorias de categorias, o que implica mais níveis de classificações (indivíduos, classes, metaclasses, metametaclasses etc)[107]. Muitos dos exemplos apresentados na Seção 2.6 apresentam três níveis de seres: nível metaconceitual, nível conceitual e nível de indivíduos. Em outras palavras, esses exemplos apresentam indivíduos que são instâncias de categorias do nível conceitual e,

por sua vez, as categorias do modelo de objetos são instâncias das metacategorias do nível metaconceitual.

Para atender situações como essa, pesquisadores do Núcleo de Estudos em Modelagem Conceitual e Ontologias (NEMO)[25] desenvolveram uma teoria multi-nível de classificação conhecida por *Multi-Level Theory* (MLT). Detalhes sobre a MLT são apresentados em Almeida et al. [107] e em Carvalho e Almeida [108].

A MLT propõe as seguintes definições [107, tradução nossa, grifos dos autores]:

- *Indivíduos* são aquelas entidades que provavelmente não podem exercer o papel de um tipo numa relação de instanciação. Exemplos incluem ALBERT EINSTEIN, O LEÃO CECIL e o TERREMOTO DE 1985 NA CIDADE DO MÉXICO;
- *Tipos de primeira ordem* são aqueles tipos cujas instâncias são indivíduos. Exemplos incluem PESSOA, ADULTO, LEÃO, ANIMAL, ORGANISMO, TERREMOTO, DESASTRE NATURAL, EVENTO;
- *Tipos de segunda ordem* são aqueles tipos cujas instâncias são tipos de primeira ordem. Exemplos incluem TIPO DE PESSOA POR IDADE, ESPÉCIES DE ANIMAIS, ESPÉCIES, TIPO DE EVENTO;
- *Tipos de terceira ordem* são aqueles tipos cujas instâncias são tipos de segunda ordem (CLASSIFICAÇÃO TAXONÔMICA), e assim por diante. A ordem mais alta pode ser estabelecida conforme requisitos da aplicação, e o esquema pode ser estendido para lidar com um número arbitrário de níveis[26].

Os autores também declaram que uma *entidade* é um indivíduo ou uma categoria e que dois tipos são iguais se todas as suas instâncias possíveis forem as mesmas.

Além disso, Almeida et al. definem as seguintes relações:

- *Especialização*: quando um tipo α especializa um tipo β implica que todas as instâncias de α são também instâncias de β;
- *Especialização própria*: a extensão do tipo especializado é um subconjunto próprio da extensão do tipo mais geral;

[25] Grupo de estudos em modelagem conceitual e ontologias vinculado à Universidade Federal do Espírito Santo (UFES)

[26]
- Individuals *are those entities which cannot possibly play the role of type in the instantiation relation. Examples include ALBERT EINSTEIN, CECIL THE LION and the 1985 MEXICO CITY EARTHQUAKE.*
- First-order *types are those types whose instances are individuals. Examples include PERSON, ADULT, LION, ANIMAL, ORGANISM, EARTHQUAKE, NATURAL DISASTER, EVENT.*
- Second-order types *are those types whose instances are first-order types. Examples include PERSON TYPE BY AGE, ANIMAL SPECIES, SPECIES, EVENT TYPE.*
- Third-order types *are those types whose instances are second-order types (TAXONOMIC RANK), and so on. The topmost order can be established as required by applications, and the scheme can thus be extended to cope with an arbitrary number of levels.*

- *Powertype*: um tipo δ é *powertype* de um tipo η se e somente se todas as instâncias de δ são especializações de η e todas as especializações possíveis de η são instâncias de δ;
- *Categorização*: uma relação de categorização é similar à noção de *powertype* apresentada, porém o tipo base η é excluído do conjunto de instâncias do powertype e nem todas as especializações de η precisam ser instâncias do *powertype*;
- "Especializações da relação de categorização são definidas de modo a capturar cenários distintos de categorização: *categorização disjunta*, para acomodar os casos em que cada instância do tipo base é instância de pelo menos uma instância do tipo de mais alta ordem; *categorização completa*, em que cada instância de um tipo base é instância de pelo menos uma instância do tipo de mais alta ordem; e *segmentação*, quando uma instância do tipo base é instância de exatamente uma instância do tipo de mais alta ordem[27] [107, tradução nossa, grifos dos autores]."

Da MLT, derivam-se as seguintes regras que serão utilizadas neste trabalho [107]:

1. O relacionamento de *instanciação* é irreflexível, antissimétrico e antitransitivo. Também só pode existir entre entidades de níveis adjacentes;
2. O relacionamento de *especialização* é reflexivo, antissimétrico e transitivo;
3. O relacionamento de *especialização própria* é irreflexível, antissimétrico e transitivo;
4. As duas relações de especialização só podem acontecer entre indivíduos de mesmo nível;
5. Os relacionamentos *é powertype de* e *caracteriza* somente podem existir entre níveis adjacentes de entidades. Em outras palavras, o tipo base deve ser uma ordem de grandeza inferior ao tipo de ordem mais alta;
6. O *powertype* de um tipo base é único para aquele tipo base;
7. Tipos que categorizam o tipo base sempre especializam o powertype do tipo base;
8. Se dois tipos particionam o mesmo tipo base, eles não podem especializar um ao outro.

É possível combinar MLT com UFO, conforme apresentado na Figura 2.20. A integração entre MLT e UFO é discutida em detalhes em Carvalho et al. [109].

[27] *Specializations of the* categorization *relation were defined in order to capture different scenarios of categorization:* disjoint categorization*, to accommodate the cases in which each instance of the base type is instance of at most one instance of the higher-order type;* complete categorization*, when each instance of the base type is instance of at least one instance of the higher-order type; and* partitioning*, when an instance of the base type is instance of exactly one instance of the higher-order type*

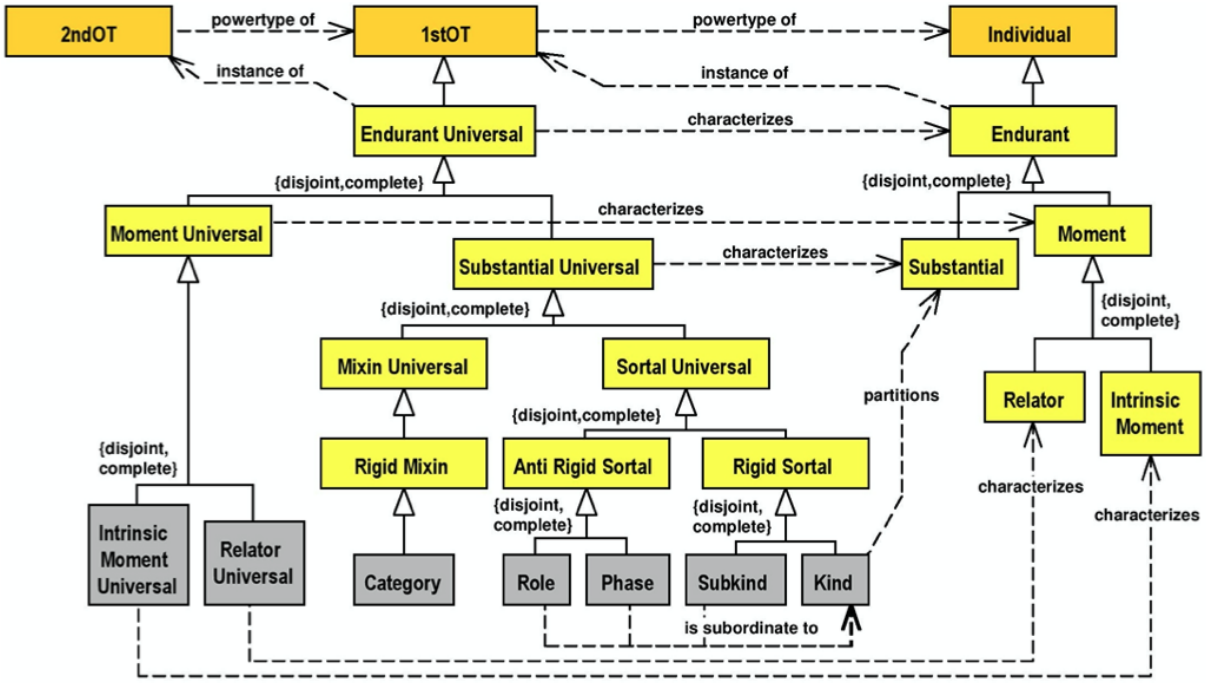


Figura 2.20: Aplicação de MLT para a taxonomia de endurantes da UFO. Fonte: Carvalho et al. [109]

2.10 Trabalhos Correlatos

Rybola [110] fez um mapeamento de OntoUML[28] para banco de dados relacionais baseada no paradigma *Model Driven Development* (MDD). Rybola apresenta a transformação de um *Platform Independent Model* (PIM) (modelo independente de plataforma) OntoUML para um PIM UML, preservando as restrições semânticas e específicas de domínio de diversos universais e tipos de relacionamentos definidos no modelo OntoUML. O autor ainda transforma o UML PIM em um *Platform Specific Model* (PSM) (modelo específico de plataforma) para *Relational Database* (RDB) (bancos de dados relacionais), com foco na implementação das restrições derivadas do OntoUML PIM de modo a preservar restrições. Por fim, ele propõe transformar o RDB PSM em um SQL *Implementation Specific Model* (ISM), com foco em algumas implementações de restrições derivadas do OntoUML PIM original.

Nossa abordagem difere de Rybola no sentido que criamos um AOM para a interpretação computacional de modelos de objetos baseados em UFO. Além disso, no trabalho de Rybola, o modelo de objetos é transformado em tabelas de banco de dados. Na nossa abordagem, as tabelas de banco de dados são fixas e o modelo de objetos é inserido nestas tabelas. Em outras palavras, no trabalho do autor, as categorias de domínio se transfor-

[28]Versão da UML ontologicamente bem fundamentada e baseada em UFO.

mam em tabelas de banco de dados. Na nossa abordagem, as categorias de domínio são instâncias das tabelas de banco de dados.

Zhang et al. [111] desenvolvem um sistema de simulação de parâmetros baseado em ontologias e AOM. Ele consiste em três camadas assim denominadas: camada do metamodelo, camada de domínio e camada de instâncias. Um interpretador é construído baseado na camada do metamodelo. Especialistas do domínio podem alterar o modelo instanciando um metamodelo usando *Protégé*. Os autores utilizam-se de banco de dados *Not Only SQL* (NoSQL) sem esquema pré-definido. O sistema se adapta modificando o modelo de domínio. O modelo de domínio existe como um arquivo de configuração independente, podendo ser alterado durante tempo de execução.

Nossa abordagem difere de Zhang et al. no sentido que se utiliza de UFO para construir um modelo de objetos ontologicamente bem fundamentado. Os autores utilizam-se de um banco de dados NoSQL para conseguir flexibilidade. Nossa abordagem utiliza-se de um banco de dados relacional com tabelas fixas baseadas no quadrado aristotélico e no dodecágono ontológico, ou seja, a flexibilidade da nossa abordagem se vale do fato do modelo de objetos ser transformado em instâncias das tabelas de banco de dados.

2.11 Considerações Finais

Este capítulo discute conceitos e detalhes importantes para a compreensão da pesquisa, como modelos, metamodelagem, modelagem conceitual, modelos adaptativos de objetos, ontologias, Ontologia, o quadrado aristotélico, o dodecágono ontológico, MLT e UFO.

Capítulo 3

Arquitetura e Modelos Construídos

Este Capítulo apresenta a descrição da arquitetura proposta como solução para o problema de pesquisa.

Conforme descrito na Seção 1.1, modelar conceitualmente uma solução para domínios complexos é uma atividade com muitas incertezas, propensa a alterações constantes. Um exemplo de domínio complexo é o domínio de gestão de normas jurídicas, pois normas jurídicas tratam de vasta amplitude de seres que compõem o mundo físico e social. Modelar todas as entidades do domínio antecipadamente não é viável.

Para abordar o problema, este trabalho tem como objetivo desenvolver um modelo adaptativo de objetos para interpretar computacionalmente modelos de objetos baseados em UFO (vide Subseção 1.3.1). Esta abordagem deve permitir a alteração de um modelo de objetos que estende a UFO sem necessidade de alteração de código nem do modelo de aplicação. Para isso, este trabalho seguiu as seguintes premissas:

1. O dodecágono ontológico é suficiente para representar as categorias da UFO-A;
2. A UFO é suficiente para a produção de modelos de objetos sólidos;
3. É possível interpretar as instâncias das categorias do modelo de objetos e produzir um sistema de informação descritivo.

Este capítulo apresenta uma arquitetura AOM com um modelo de objetos baseado na UFO e um modelo de aplicação inspirado no quadrado aristotélico (Seção 2.3) e no dodecágono ontológico (Seção 2.4). O modelo de aplicação é capaz de interpretar computacionalmente o modelo de objetos. O modelo de aplicação apresenta, também, extensões para tratar eventos ontológicos e epistemológicos.

O modelo de objetos é apresentado na Seção 3.1 enquanto o modelo de aplicação, na Seção 3.2.

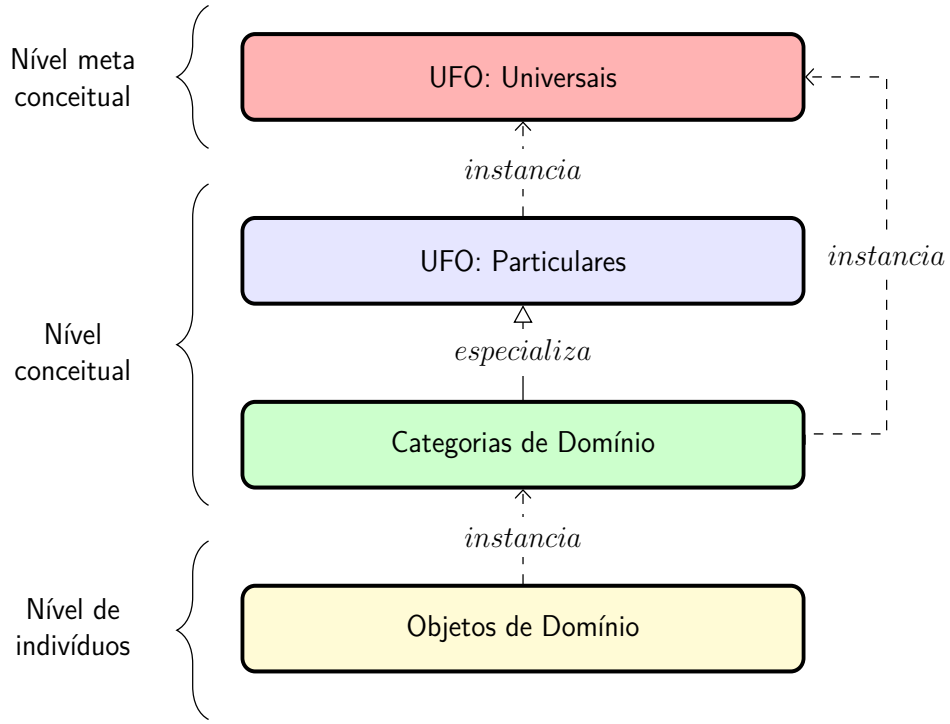


Figura 3.1: Níveis de entidades do modelo de objetos (Fonte: Os autores)

3.1 Modelo de objetos

Apresentam-se as entidades do modelo de objetos de acordo com o mesmo critério utilizado na Seção 2.6 e em Zamborlini [67], ou seja, as entidades UFO e de domínio são divididas em três níveis disjuntos:

1. nível *meta-conceitual*, que contém as categorias de universais da UFO;
2. nível *conceitual*, que contém as categorias de particulares da UFO e as categorias de domínio. As categorias de domínio são especializações das categorias de particulares da UFO e instâncias das categorias de universais; e
3. nível de *indivíduos*, que contém os objetos de domínio.

Detalham-se nas Subseções 3.1.1 a 3.1.4, as entidades que compõem os três níveis.

3.1.1 Nível meta-conceitual: Universais da UFO

O nível meta-conceitual contém um fragmento dos universais da UFO, mais especificamente, algumas das categorias de Universais da UFO-A acrescidas pela categoria *Event Universal* da UFO-B. O fragmento de *Universais* utilizado é exibido na Figura 3.2.

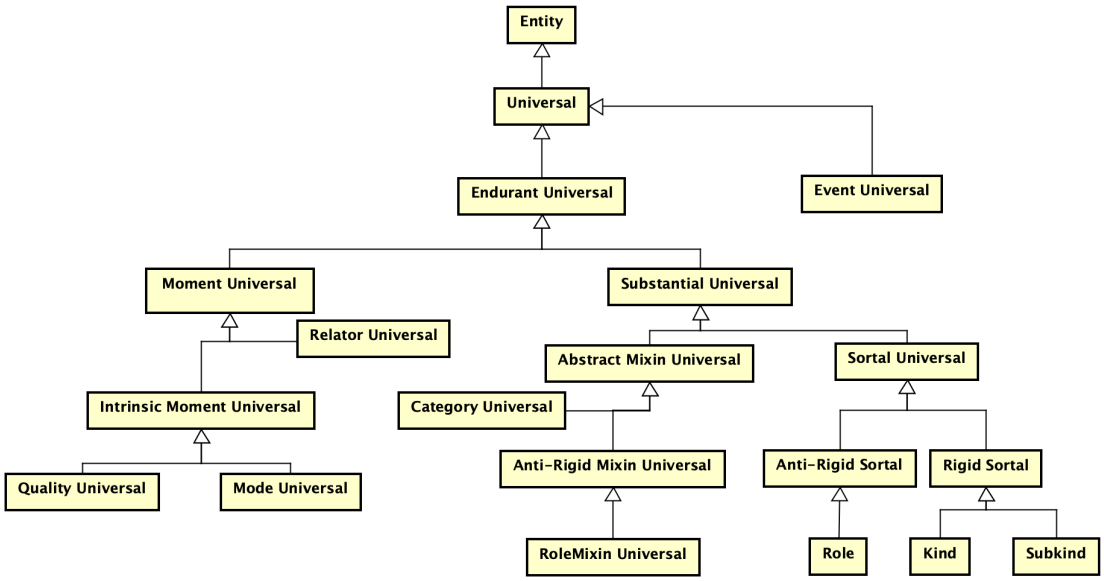


Figura 3.2: Fragmento das Categorias de Universais da UFO.

3.1.2 Nível Conceitual: Particulares da UFO

As categorias de particulares da UFO encontram-se no nível conceitual do modelo de objetos. A Figura 3.3 apresenta as categorias de Particulares da UFO utilizadas no contexto deste trabalho.

3.1.3 Nível Conceitual: Categorias de Domínio

As categorias de domínio pertencem ao nível conceitual do modelo de objetos. Cada categoria de domínio deve, obrigatoriamente, ser:

1. especialização, direta ou indireta, de uma categoria da taxonomia de *particulares* da ontologia de fundamentação. Uma categoria de domínio pode especializar múltiplas categorias de domínio; e
2. instância direta de uma, e somente uma, categoria de *universais* da ontologia de fundamentação.

As Figuras 2.7, 2.9 e 2.13 explicitam as relações entre categorias de domínio e as categorias UFO, ou seja, a relação de especialização com as categorias de particulares e a relação de instanciação com as categorias de universais da UFO. Já as Figuras 3.4 e 3.5 apresentam alguns exemplos de categorias de domínio em notação OntoUML.

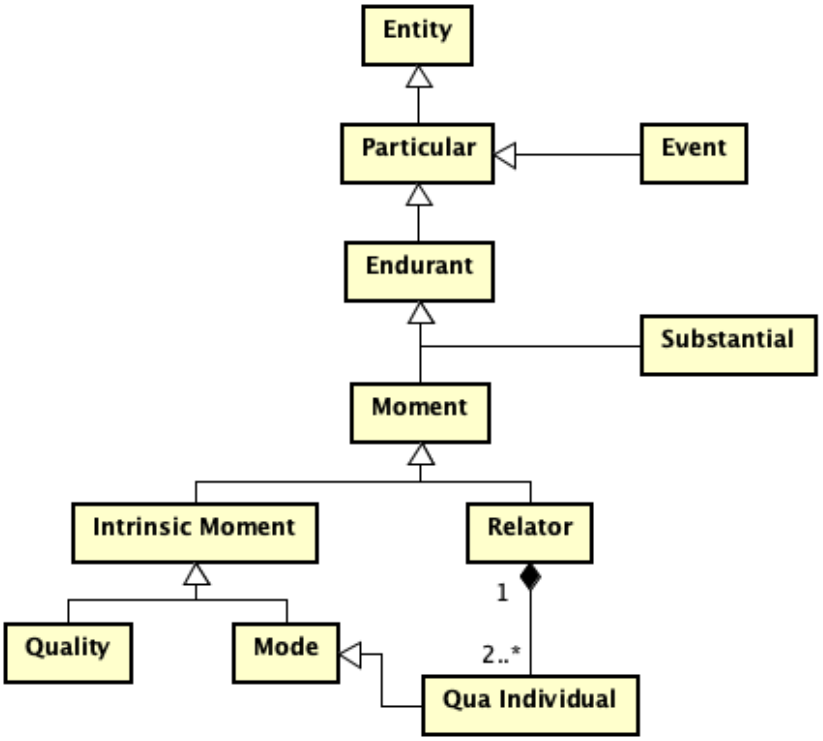


Figura 3.3: Fragmento da Categoria de Particulares da UFO

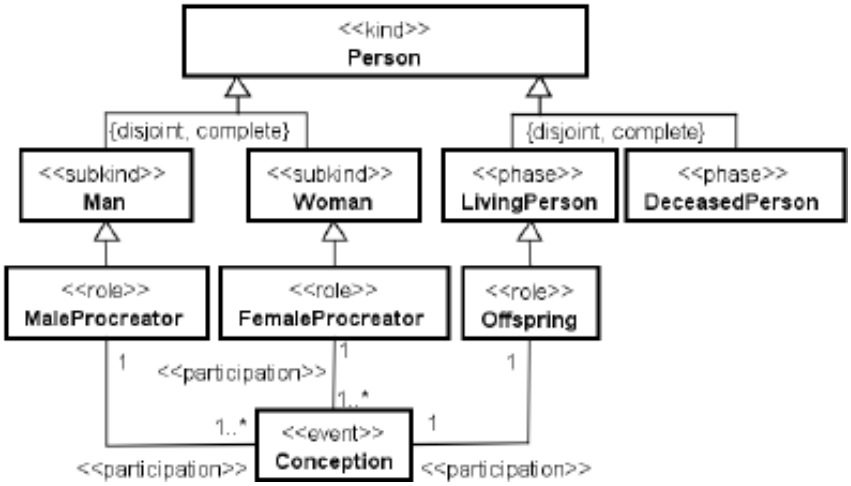


Figura 3.4: Exemplo simples de categorias do domínio de genealogias. Fonte: Carvalho et al. [112]

3.1.4 Nível de indivíduos: objetos de domínio

O nível de indivíduos contém os objetos de domínio. Os objetos de domínio são instâncias das categorias de domínio (Figura 3.1) e representam objetos e eventos que existem na

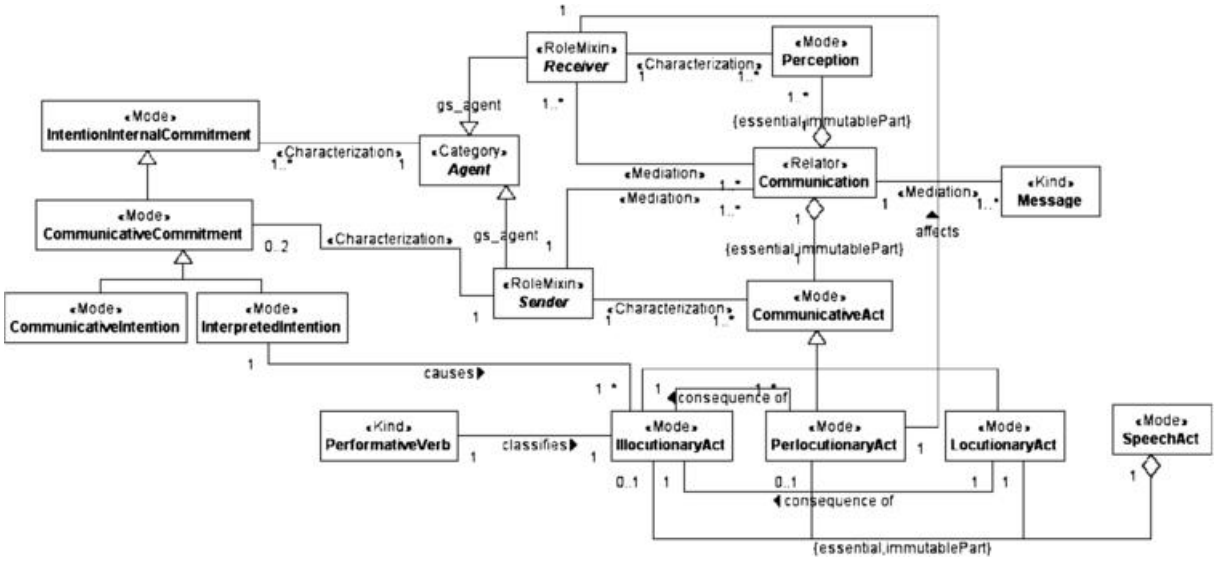


Figura 3.5: Fragmento com conceitos de uma ontologia de comunicação. Fonte: Morales-Ramirez et al. [113]

realidade. Alguns exemplos:

- *Juscelino Kubitschek* e *Novak Djokovic*, instâncias da categoria *Pessoa*;
- *Matrícula de João na UnB*, instância da categoria *Matrícula*;
- *Lei 8.112 de 1990*, instância da categoria *Lei Ordinária*;
- *Nascimento de Sigmund Freud*, instância de *Evento*;
- *São Paulo*, instância de *Cidade*.

Portanto, na arquitetura AOM proposta neste manuscrito, o modelo de objetos contém entidades dependentes de domínio e entidades UFO. O modelo de aplicação deve, portanto, ser capaz de interpretar entidades de nível meta-conceitual, conceitual e de indivíduos. O modelo de aplicação proposto é apresentado na Seção 3.2.

3.2 Modelo de aplicação

Para permitir a representação das entidades apresentadas na Seção 3.1, propõe-se um modelo de aplicação baseado na ontologia de quatro categorias de Lowe (Seção 2.3) e no dodecágono ontológico (Seção 2.4) e que, com algumas extensões, seja capaz de representar eventos, aspectos epistemológicos e ontológicos.

O modelo de aplicação é composto de duas camadas:

1. uma camada física, que representa o banco de dados do modelo de aplicação;

2. uma camada lógica, que demonstra como é feito o mapeamento do modelo de objetos para o banco de dados.

A camada lógica, que convencionamos chamar de $\mathcal{L}1$, é representada com diagramas UML. Já a camada física de banco de dados, utiliza-se da notação conhecida como pé-de-galinha (*crow's foot*) [114]. A Figura 3.6 resume a notação pé-de-galinha utilizada neste texto. Especial atenção deve ser dada às seguintes características:

- chaves primárias contém o estereótipo *PK* antes do nome das colunas que compõem a chave primária. Além disso, as chaves primárias são sublinhadas;
- chaves estrangeiras contém o estereótipo *FK* antes do nome da coluna com chave estrangeira;
- colunas de preenchimento obrigatório são destacados em negrito;
- cada lado de um relacionamento entre tabelas possui cardinalidade conforme descrito na Figura 3.6

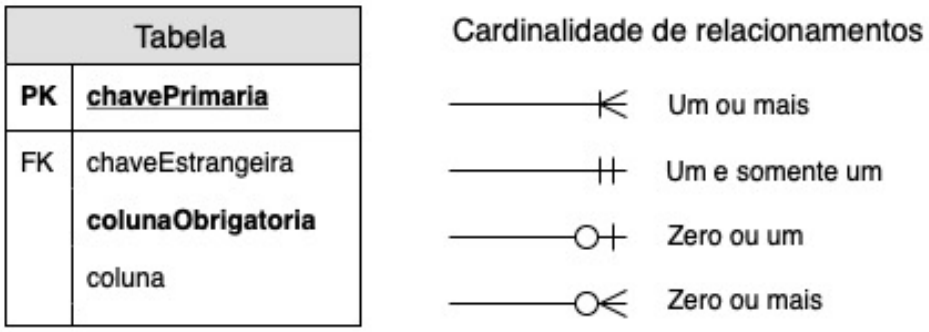


Figura 3.6: Notação dos diagramas físicos de banco de dados

O modelo de aplicação é essencial para permitir a evolução da solução sem alteração de código-fonte ou criação/alteração de tabelas de banco de dados quando novas propriedades ou categorias são modeladas, inseridas ou modificadas. Ele permite gerenciar objetos transversalmente ao gerenciamento de suas propriedades. Quando houver necessidade de acrescentar uma propriedade a uma categoria de objeto, a propriedade pode ser adicionada ao modelo de objetos em tempo de execução. Para isso, necessita-se apenas de inclusões de tuplas ao banco de dados. De modo semelhante, quando uma nova categoria é identificada, basta a inclusão desta categoria no modelo de objetos. Nenhuma alteração de código é necessária e o sistema se adapta às novas categorias e propriedades.

Como apresenta a Figura 3.7, o núcleo do modelo de aplicação proposto baseia-se na ontologia de quatro categorias aristotélicas (Seção 2.3), sendo constituído por quatro tabelas: *Type*, *Entity*, *PropertyType* e *Property*.

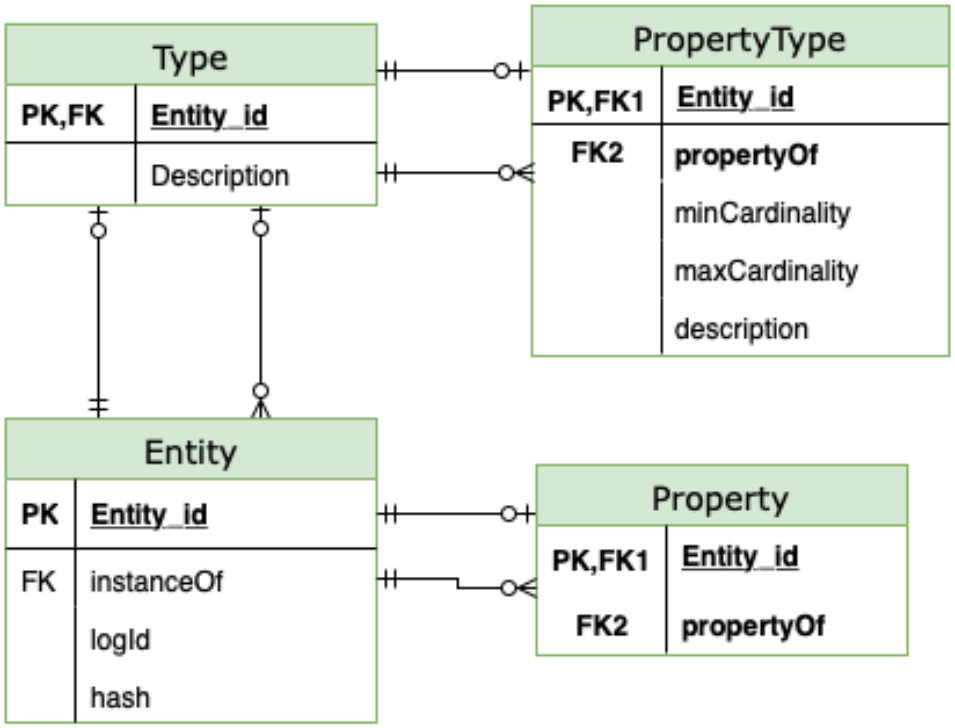


Figura 3.7: Núcleo do modelo de banco de dados

Apesar de inspirado no quadrado aristotélico, as categorias do núcleo do modelo físico de banco de dados diferem do quadrado aristotélico no seguinte sentido:

1. A tabela *Type* é mais abrangente que a categoria de *Substantial Kinds* do quadrado. No quadrado aristotélico, as quatro categorias de entidades são disjuntas. Logo, *Substantial Kinds* contém apenas tipos substanciais. Entretanto, a tabela *Type* representa quaisquer *tipos*, não apenas de *substanciais*. Logo, tipos de não-substanciais, como *categorias de Relators* e *categorias de momentos*, são representados nesta tabela. Outra diferença, é que a tabela também representa *tipos* de qualquer ordem, do ponto de vista da MLT (vide Seção 2.9). Portanto, tanto as categorias de nível metaconceitual quanto do nível conceitual do modelo de objetos são representadas nesta tabela. Outra distinção em relação ao quadrado, é que a tabela representa também as categorias de *perdurantes*. Em suma, a tabela *Type* representa todos os tipos do modelo de objetos, independente de serem substanciais, não substanciais, do nível a que pertencem e de serem perdurantes ou endurantes;

2. A tabela *Entity* representa todas as entidades do modelo de objetos, não apenas a categoria de *Individual Substances* do quadrado: todo *tipo*, de qualquer nível, *substancial* ou não e todo *objeto do domínio* são representados nesta tabela;

3. A tabela *PropertyType*, por outro lado, é mais restrita que a categoria de *Attributes* do quadrado ontológico. Ela representa apenas atributos que sejam intrínsecos a algum tipo presente na tabela *Type*. Como consequência disso, esta tabela representa as categorias UFO de *qualidades* e *modos*, mas não representa *Relators*;

4. A tabela *Property*, de modo similar, é mais restrita que a categoria de *Modes* do quadrado aristotélico. Ela representa apenas instâncias das categorias da tabela *PropertyType*.

Para mapear entidades do modelo de objetos para as quatro tabelas que formam o núcleo do banco de dados, utiliza-se as entidades do modelo $\mathcal{L}1$ da Figura 3.8.

Importante destacar que os estereótipos dos diagramas UML que contém informação de mapeamento para $\mathcal{L}1$ e para banco de dados significam *mapeamento*. Eles *não* representam uma relação de instanciação entre a entidade do diagrama e a entidade do estereótipo. Por mapeamento, entende-se que instâncias da entidade do diagrama será instância da entidade do estereótipo. Para evitar confusões, ao abordar entidades de modelos diferentes neste texto, utiliza-se o prefixo $\mathcal{L}1$ antes das entidades do modelo lógico e o prefixo *DB* antes das entidades do banco de dados.

Além disso, os estereótipos apresentados em cada categoria de $\mathcal{L}1$ representam a tabela mais específica em que a entidade do diagrama é representada. Por isso, a entidade $\mathcal{L}1$*:PropertyType* possui o estereótipo $\ll$ `DB:PropertyType` $\gg$ indicando que *DB:PropertyType* é a tabela mais específica em que $\mathcal{L}1$*:PropertyType* é representada. Como *DB:PropertyType* especializa *DB:Type* e *DB:Type*, por sua vez, especializa *DB:Entity*, então uma entidade registrada em *DB:PropertyType* será registrada em *DB:Type* e em *DB:Entity*. Em outras palavras, a chave primária de *DB:PropertyType* é chave estrangeira que aponta para a chave primária de *DB:Type*. Do mesmo modo, a chave primária de *DB:Type* é chave estrangeira que aponta para a chave primária da tabela *DB:Entity*.

Como apresentado em Guizzardi e Wagner [89] e na Seção 2.4, apenas as quatro categorias do quadrado aristotélico não são suficientes para descrever quaisquer entidades do mundo social ou físico.

Por isso, este trabalho apresenta quatro extensões ao núcleo do modelo lógico:

1. a primeira extensão visa permitir tratar as doze categorias do dodecágono ontológico;

2. a segunda extensão possibilita a representação da relação de especialização entre categorias;

3. a terceira extensão permite indexar instâncias no tempo;

4. a quarta extensão divide os eventos que alteram o estado do banco de dados em duas categorias:

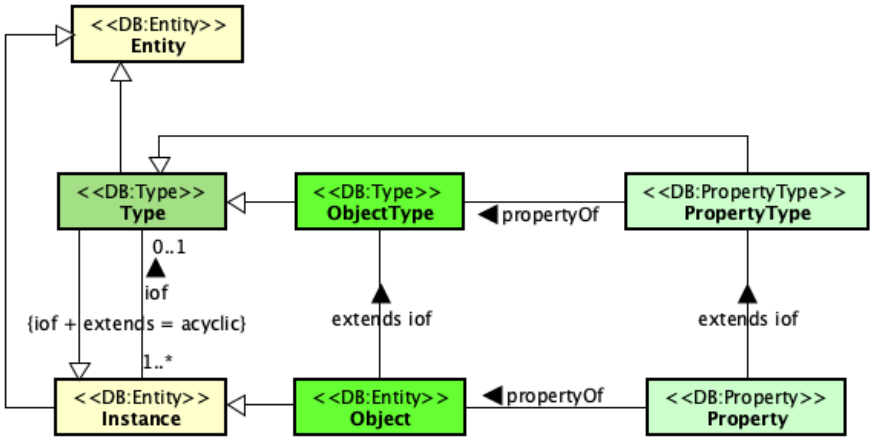


Figura 3.8: Núcleo do modelo lógico em $\mathcal{L}1$

- eventos de mundo, ou ontológicos;
- eventos de sistema, ou epistemológicos.

As subseções seguintes apresentam estas extensões em detalhes.

3.2.1 Adequação do modelo ao dodecágono ontológico

A Seção 2.4 apresenta o dodecágono ontológico de Guizzardi e Wagner (Figura 2.2) como uma extensão da ontologia de oito categorias da OWL (Figura 2.3) e do quadrado aristotélico (Seção 2.3). Guizzardi e Wagner [89] defendem a necessidade de doze categorias para correta modelagem conceitual de endurantes: seis categorias de universais acrescidas das seis categorias de particulares que as instanciam.

Consoante ao exposto na Seção 2.4, para estender a ontologia de quatro categorias, a ontologia da OWL divide a categoria de *Attribute/Moments Type* em duas categorias distintas: *Attribute* e *Relationship type*. *Attributes* são "tipos de relações binárias (chamadas de *data properties* em OWL), cujo contra-domínio é um tipo de dado (*Data type*)"[1]. *Relationship types* são "tipos de relações binárias cujo contra-domínio é um tipo de objeto *Object type*"[2][89, p. 2, tradução nossa]. Como *Attributes* estão relacionados a um tipo de dado e tipos de dado não são tipos de objeto, necessita-se da categoria de *Data types*. Para chegar às doze categorias do dodecágono, Guizzardi e Wagner [89] defendem a utilização de mais duas categorias: *quality type*, que fundamenta *attribute*, e *relator type*, que fundamenta *relationship type*.

[1] *[...] are binary relationship types (called 'data properties' in OWL) where the range is a datatype.*
[2] *[...] are binary relationship types [...] where the range is an object type.*

As cinco categorias mencionadas são universais. Existem ainda cinco categorias de particulares que instanciam cada um dos universais acima:

- *Data value*, instância de *data type*;
- *Attribution*, instância de *attribute*;
- *Quality*, instância de *quality type*;
- *Relator*, instância de *relator type*;
- *Relationship*, instância de *relationship type*.

Portanto, o dodecágono é constituído por essas dez categorias e pelas categorias *Object type* e *Object*, já presentes no quadrado aristotélico (vide Figura 3.9).

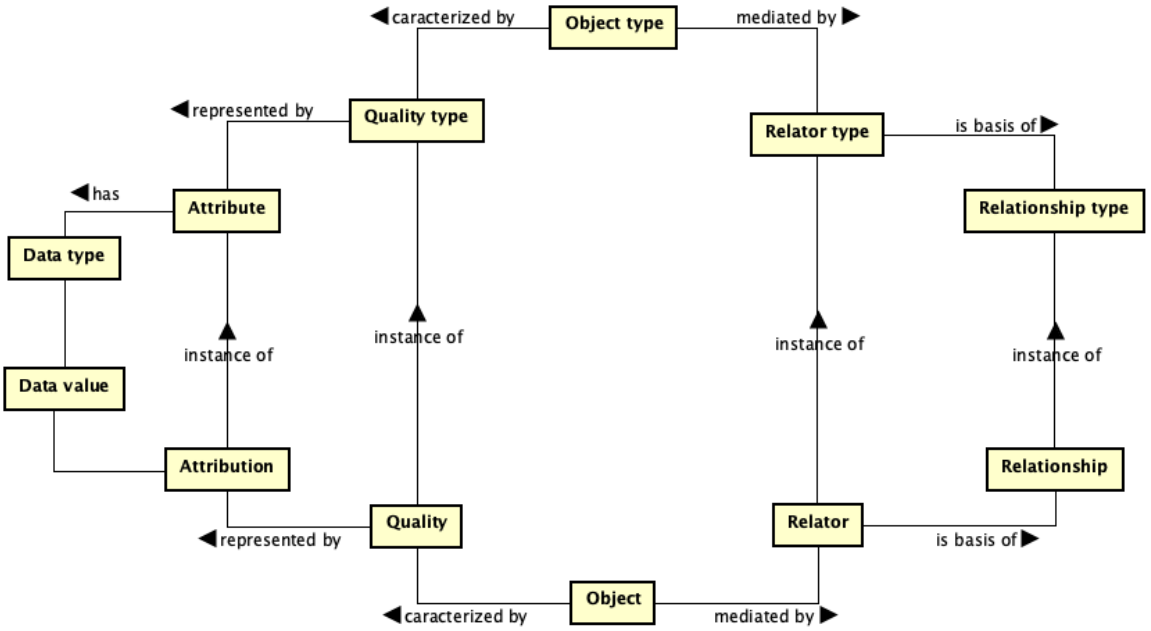


Figura 3.9: Dodecágono ontológico. Fonte: Adaptado de Guizzardi e Wagner [89]

Representação de *quality types, attributes, qualities* e *attributions* em $\mathcal{L}1$

Em $\mathcal{L}1$ utiliza-se de uma abordagem similar ao dodecágono ontológico para representar entidades. $\mathcal{L}1$, conforme apresenta a Figura 3.10, divide os tipos de propriedades (*moments*) em duas categorias:

- propriedades relacionais (*relational properties*), cujo contradomínio são tipos de objetos;

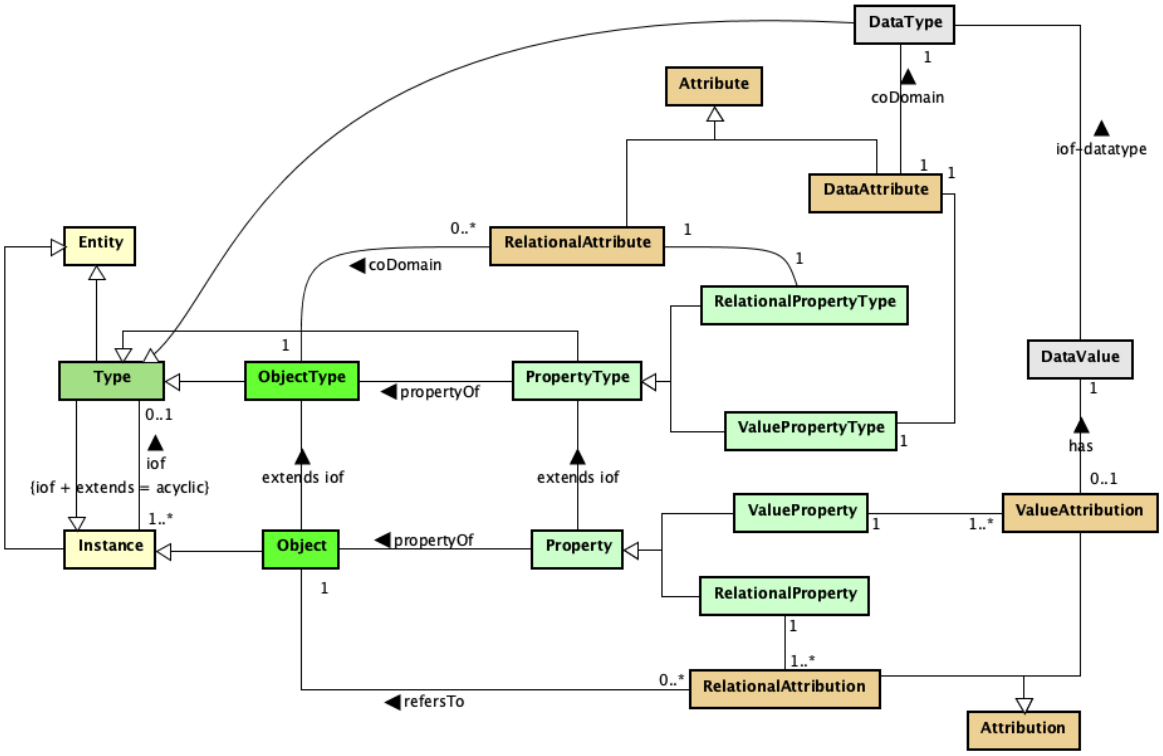


Figura 3.10: Extensão ao modelo lógico $\mathcal{L}1$ para representar o dodecágono ontológico

- propriedades próprias (*data properties*), cujo contradomínio são tipos de dados.

No dodecágono ontológico, um *quality type* caracteriza um *object type* (Figura 2.2) enquanto um *quality* instancia um *quality type* e é inerente a um *object*. De modo similar, em $\mathcal{L}1$, $\mathcal{L}1$*:PropertyType* é uma propriedade de $\mathcal{L}1$*:ObjectType* e $\mathcal{L}1$*:Property* é uma propriedade de $\mathcal{L}1$*:Object*.

Assim, o *quality type cor de olho humano* caracteriza apenas o *object type humano* enquanto o *quality cor dos olhos de Maria* é inerente unicamente a Maria e sempre a Maria. Outrossim, um $\mathcal{L}1$*:PropertyType* é propriedade de apenas um $\mathcal{L}1$*:ObjectType* e um $\mathcal{L}1$*:Property* é propriedade sempre do mesmo e único $\mathcal{L}1$*:Object*.

No dodecágono, *Attributes* são relações binárias cujo contra-domínio é um tipo de dado. Um *attribute* relaciona um *quality type* com seu respectivo contradomínio (tipo de dados). Em $\mathcal{L}1$, a categoria do dodecágono *attributes* é representada em $\mathcal{L}1$*:DataAttribute*. Em outras palavras, $\mathcal{L}1$*:DataAttribute* é a ligação entre $\mathcal{L}1$*:ValueProperty* e seu $\mathcal{L}1$*:DataType*. De modo semelhante, $\mathcal{L}1$*:ValueAttribution* conecta um *quality* a um valor.

Representação de *relator types, relators, relationship types* e *relationships* em $\mathcal{L}1$

Por outro lado, *relators* não são explicitamente representados como uma categoria separada em $\mathcal{L}1$, mas de forma indireta, por meio da noção de *qua indivíduo* apresentada na Subseção 2.6.6. Como *Relators* são a agregação dos *qua indivíduos* que o formam, este trabalho opta por representar *Relators* e *Relationships* da seguinte forma:

- os *Relators* são representados em $\mathcal{L}1$ como $\mathcal{L}1$*:Object*. *Relator Type*, por sua vez, é representado como $\mathcal{L}1$*:ObjectType*;
- os *qua indivíduos* de um *relator* são representadas como $\mathcal{L}1$*:Property*. Os *qua indivíduos* são instâncias dos *qua indivíduos types* representados em $\mathcal{L}1$*:PropertyType*;
- em $\mathcal{L}1$*:RelationalAttribution*, qua indivíduos referem-se a um $\mathcal{L}1$*:Object*. De modo similar, em $\mathcal{L}1$*:RelationalAttribute* é possível representar o contra-domínio de $\mathcal{L}1$*:RelationalPropertyType*, que é um $\mathcal{L}1$*:ObjectType*. Nos casos de relações materiais, é necessária uma restrição para garantir que o contra-domínio seja o conjunto de *relator types*. Um *qua indivíduo* deve ser inerente a um único objeto e referenciar um único *relator*;
- um *relator universal* deve estar relacionado a pelo menos dois *qua indivíduos*. Por isso, o *relator* é representado como a soma dos *qua indivíduos* que o referenciam;
- um *relator* representa pelo menos dois *relacionamentos*. Por exemplo, o relator *Casamento de João e Maria* possui os relacionamentos: *João é marido de Maria* e *Maria é esposa de João*. Cada relacionamento é obtido por meio do par $\mathcal{L}1$*:RelationalProperty*/$\mathcal{L}1$*:RelationalAttribution*.

Representação de Relações Formais em $\mathcal{L}1$

Segundo a teoria original de Guizzardi, apenas *relações materiais* são mediadas por *relators*. Entretanto, Guarino et al. [115] defendem casos de relações formais que admitem *relators*.

Guarino et al. defendem que[3]:

[3] despite comparative relations were considered as formal in Guizzardi's sense, and therefore not deserving reification, there may be good reasons to talk of them [5], and therefore reify them: for instance, one may want to keep track of the difference in height between a mother and her son, or of the temperature difference between two bodies. So, comparative relations seem to share something in common with the other relations that deserve to be reified. According to Guarino and Guizzardi, this commonality lies in the fact that they are both descriptive relations, which hold in virtue of some particular aspects inhering in the relata5. So, their new proposal (with respect to the original Guizzardi's work) is that it is the mereological sum of these aspects that acts as relator, accounting both for the fact that the relation holds and for the way the relata are linked together, which may vary in time. Under this view, the relator of

> ... apesar de relações comparativas serem consideradas formais no sentido de Guizzardi e, portanto, não merecerem reificação, pode existir boas razões para falar delas [116] e, portanto, reificá-las: por exemplo, pode-se querer acompanhar a diferença de altura entre mãe e filho, ou a diferença de temperatura entre dois corpos. Assim, as relações comparativas parecem compartilhar algo em comum com as outras relações que merecem ser reificadas. Segundo Guarino e Guizzardi, essa semelhança reside no fato de serem ambas relações descritivas, que são válidas em virtude de alguns aspectos particulares inerentes aos *relata*[4]. Portanto, sua nova proposta (com relação ao trabalho original de Guizzardi) é que é a soma mereológica desses aspectos atuam como *relators*, explicando tanto o fato que a relação é mantida quanto a maneira como as *relata* são conectadas, o que pode variar no tempo. Sob essa visão, o *relator* de um relacionamento *mais alto que* é a soma das alturas (qualidades individuais) das duas *relata*, enquanto em uma relação casamento, o *relator* é uma soma de *modos dependentes externamente*, correspondendo aos compromissos e obrigações mútuos inerentes ao dois parceiros.

Portanto, de acordo com essa nova visão de Guarino et al., relações formais podem ser explicitamente reificadas por meio de um *relator* e representadas no modelo da Figura 3.12.

A Figura 3.11 apresenta o mapeamento do dodecágono ontológico para o modelo lógico $\mathcal{L}1$.

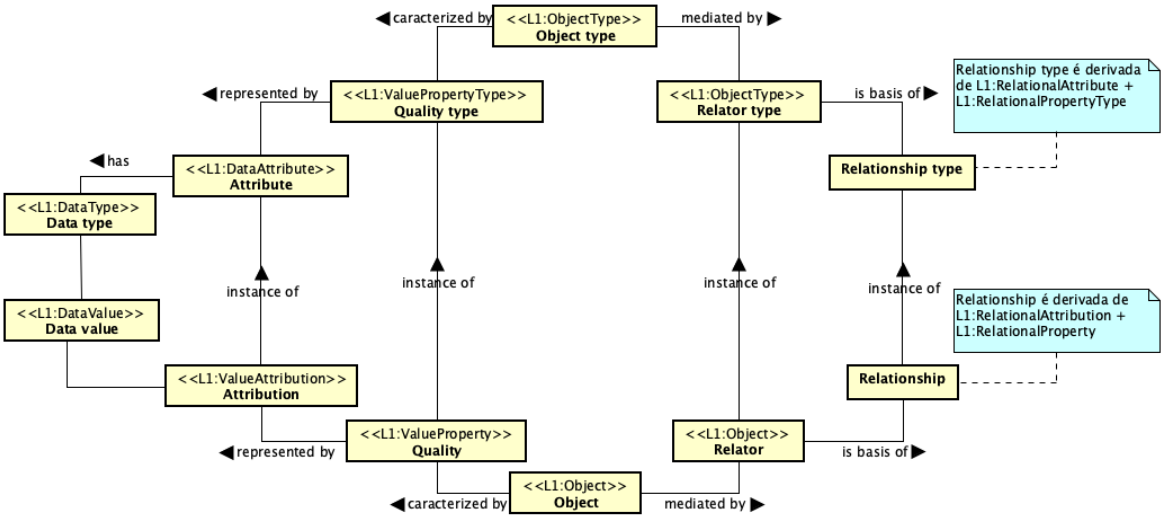


Figura 3.11: Mapeamento do Dodecágono ontológico para $\mathcal{L}1$

a taller than relationship is the sum of the heights (individual qualities) of the two relata, while for a marriage relationship the relator is a sum of externally dependent modes, corresponding to the mutual commitments and obligations inhering in the two partners.

[4][Nota do artigo original, tradução nossa]: No trabalho de Guizzardi, tais aspectos foram chamados de *moments*, e incluem *qualities* individuais e *modes* (*In Guizzardi's work, such aspects have been called moments, and include individual qualities and modes).*

Para representar as categorias de $\mathcal{L}1$ estendido, o modelo de banco de dados é acrescido de mais duas tabelas, *Attribute* e *Attribution*, conforme mostra a Figura 3.12.

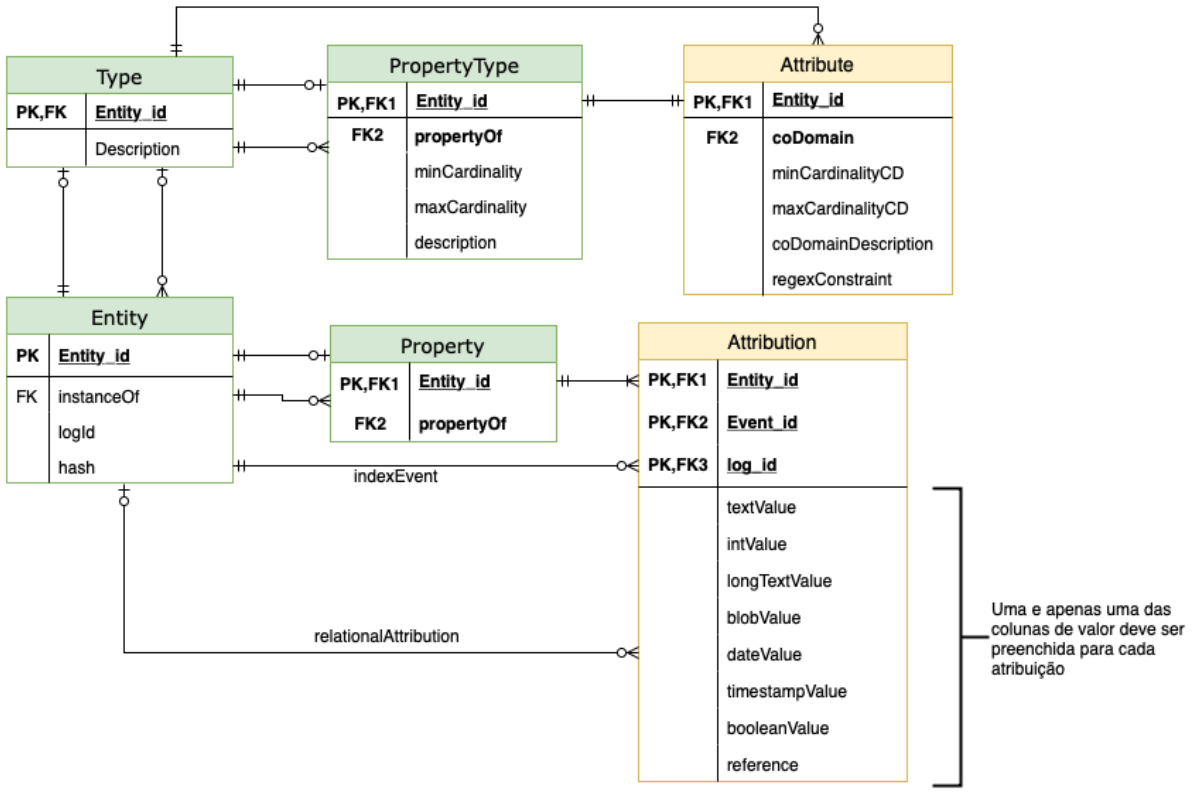


Figura 3.12: Extensão para representar $\mathcal{L}1$ com o dodecágono ontológico mapeado

Nos subtópicos seguintes, discutem-se as estratégias utilizadas para o mapeamento de $\mathcal{L}1$ para o modelo de banco de dados. A Figura 3.13 sumariza esse mapeamento.

Reificação de *Datatypes*

A categoria $\mathcal{L}1$*:DataType* foi reificada[5] e os tipos de dados possíveis registrados na tabela *DB:Type*. Um tipo de dado consiste em um espaço léxico, um espaço de valor e um mapeamento do espaço léxico para o espaço de valor [117].

Guizzardi e Wagner [89, p. 3, tradução nossa] definem um *datatype* como:

> ... uma quádrupla $\langle D, LS, VS, L2V \rangle$ em que:
>
> 1. D é o nome do *datatype*;
> 2. LS é um conjunto não vazio de caracteres Unicode chamados de espaço léxico de D. Os elementos do espaço léxico são chamados de *literais*;

[5]Reificar: "encarar (algo abstrato) como uma coisa material ou concreta; coisificar." [101, p. 1636]

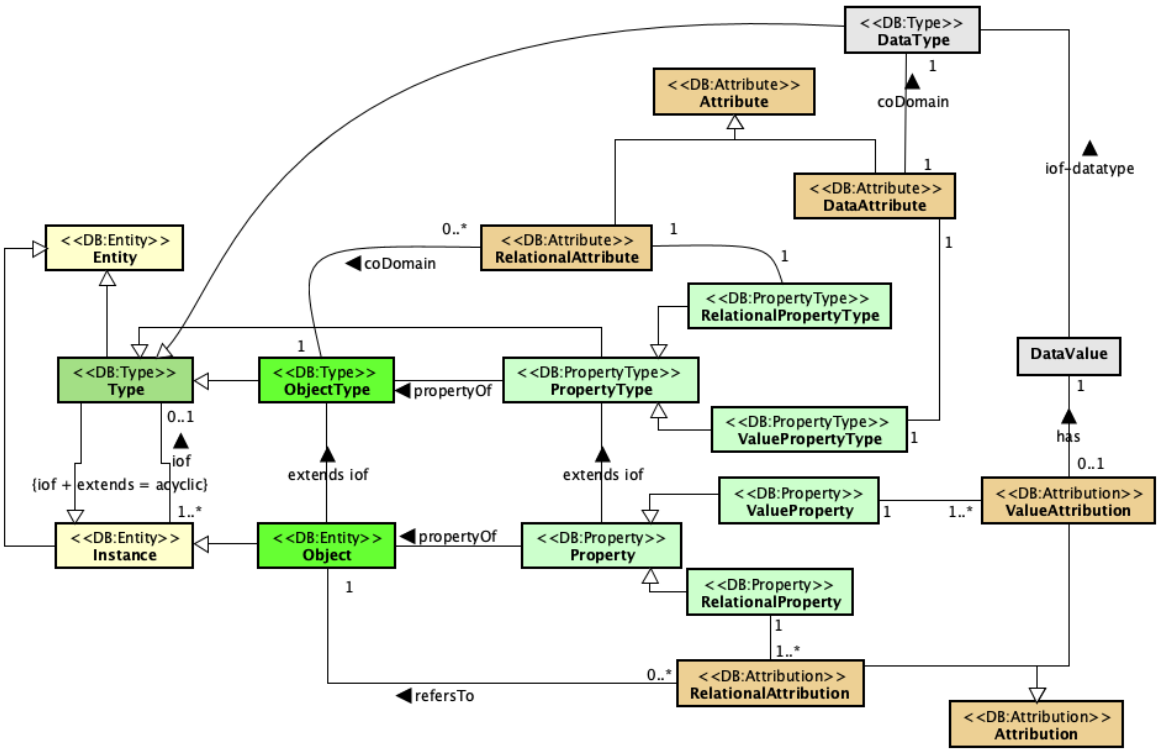


Figura 3.13: Mapeamento do modelo $\mathcal{L}1$ que contemple o dodecágono ontológico para o modelo de banco de dados

3. VS é um conjunto não vazio chamado espaço de valor de D. VS é a extensão de D e seus elementos são chamados de *valores*;
4. $L2V$ é o mapeamento de LS para VS chamado de mapeamento léxico-para-valor de D[6].

Os seguintes *datatypes* foram reificados como tuplas da tabela *DB:Type*:

1. *BinaryDatatype*: representa *objetos binários* em um banco de dados. Instâncias deste tipo são valores *Binary Large Object (BLOB)* em banco de dados;
2. *BooleanDatatype*: representa os valores *falso* ou *verdadeiro* da lógica ou da álgebra Booleana;
3. *DateDatatype*: seus valores são *strings* que representam *datas*;

[6] A datatype definition is a quadruple $\langle D, LS, VS, L2V \rangle$ where
(a) D is the name of the datatype
(b) LS is a non-empty set of Unicode character strings called the lexical space of D;
(c) VS is a non-empty set called the value space of D;
(d) $L2V$ is a mapping from LS to VS called the lexical-to-value mapping of D.

4. *DateTimeDatatype*: uma extensão ao tipo de dado *DateDatatype*, que permite representar além de datas, horários;

5. *IntegerDatatype*: os valores são números inteiros;

6. *StringDatatype*: cadeia de caracteres típica, sem marcação de estilo. Seus valores são *Variable Character Field (varchar)* em banco de dados;

7. *LongStringDatatype*: similar ao *StringDatatype*, porém permite cadeias maiores. Equivalente ao *Character Large Object* (CLOB) do banco de dados.

A relação entre os tipos de dados reificados e os tipos de dados em SQL é apresentada na Tabela 3.1.

Tabela 3.1: Relação entre tipo de dados do modelo e tipo de dados SQL

Tipo de dado no modelo	Tipo de dado em SQL
BinaryDatatype	*BLOB*
BooleanDatatype	*char*
DateDatatype	*Date*
DateTimeDatatype	*timestamp*
IntegerDatatype	*integer*
StringDatatype	*varchar*
LongStringDatatype	*CLOB*

Representação de valores de dados

As instâncias de $\mathcal{L}1$*:Datatype* são valores de dados ($\mathcal{L}1$*:DataValues*). $\mathcal{L}1$*:DataValues* são armazenados na tabela *DB:Attribution* da Figura 3.12. Esta tabela é composta de oito colunas de valores, sendo que sete são instâncias dos *datatypes* reificados. A outra coluna de valor é utilizada para referenciar objetos em propriedades referenciais e é explicada em seguida.

Representação de propriedades, atributos e atribuições em Banco de Dados

Em $\mathcal{L}1$, existem duas especializações de $\mathcal{L}1$*:PropertyType*: $\mathcal{L}1$*:RelationalPropertyType* e $\mathcal{L}1$*:ValuePropertyType*. Também existem duas especializações de $\mathcal{L}1$*:Attributes*: $\mathcal{L}1$*:RelationalAttribute* e $\mathcal{L}1$*:DataAttribute*.

Esta distinção não ocorre em banco de dados: tanto $\mathcal{L}1$*:RelationalPropertyType* quanto $\mathcal{L}1$*:ValuePropertyType* são mapeadas para *DB:PropertyType*. Semelhantemente, $\mathcal{L}1$*:RelationalAttribute* e $\mathcal{L}1$*:DataAttribute* são mapeadas em *DB:Attribute*.

Optou-se por não dividir os tipos de propriedades em mais de uma tabela, pois tanto tipos de propriedades relacionais quanto tipos de propriedades próprias têm a mesma estrutura: possuem uma chave primária[7] que indica ser aquela entidade um tipo de propriedade. Além disso, apontam para uma tupla da tabela *DB:Type*, por meio de chave estrangeira, indicando que são propriedades que caracterizam a tupla referenciada.

O mesmo acontece com $\mathcal{L}1$*:RelationalAttributes* e $\mathcal{L}1$*DataAttributes*: como $\mathcal{L}1$*:Datatypes* foram reificados na tabela *DB:Type* e $\mathcal{L}1$*:ObjectType* também são representados em *DB:Type*, então ambos os tipos de atributos possuem a mesma estrutura: são relações binárias que ligam uma propriedade ao contradomínio armazenado em *DB:Type*.

3.2.2 Representação de Especializações

Ao estender o modelo da Figura 3.7 para o representado na Figura 3.12 e o modelo da Figura 3.8 para aquele da Figura 3.10, consegue-se representar as entidades do dodecágono ontológico, entretanto não há representação para a relação de *especialização/generalização* entre tipos. De modo a permitir a representação desta relação formal, o modelo $\mathcal{L}1$ foi estendido resultando naquele da Figura 3.14. Já a expansão do modelo de banco de dados resultou na criação de mais uma tabela de generalização/especialização, conforme apresenta a Figura 3.15.

O mapeamento entre os modelos é simples: o relacionamento de especialização em $\mathcal{L}1$ é mapeado para a tabela *DB:Generalization*, conforme mostra a Figura 3.14. Esta tabela possui duas colunas:

1. *subtype*, a qual indica o tipo mais específico da relação de generalização/especialização;
2. *supertype*, a qual indica o tipo mais geral, especializado por seus subtipos.

Percebe-se que os modelos permitem especialização múltipla. Esta é uma distinção importante entre a implementação da relação de *especialização/generalização* para a relação de *instanciação* (representada por meio de chave estrangeira da tabela *Entity* para *Type* no banco de dados). Os modelos aqui propostos permitem instanciação direta de, no máximo, uma categoria.

3.2.3 Eventos ontológicos

Com as extensões apresentadas na Subseção 3.2.1 e na Subseção 3.2.2, os modelos lógico $\mathcal{L}1$ e de banco de dados passam a permitir a representação de entidades UFO-A suportadas pelo dodecágono ontológico e permite a representação de especializações entre categorias.

[7] que também é uma chave estrangeira para a tabela *DB:Type*

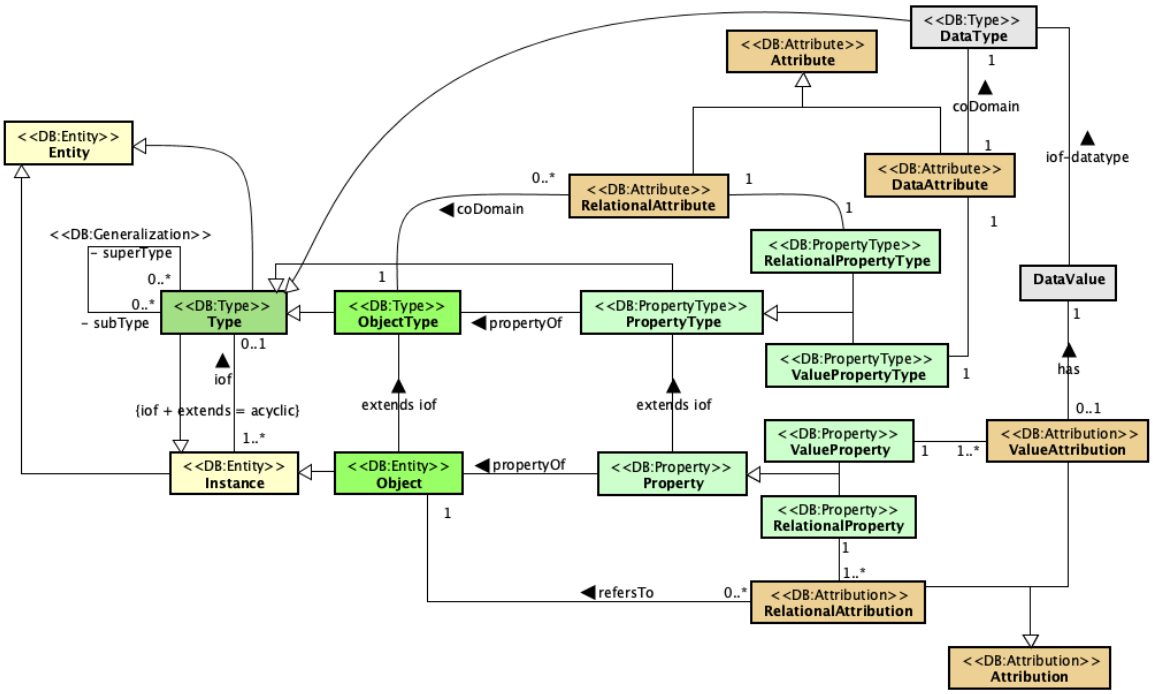


Figura 3.14: Extensão para a representação de generalizações em $\mathcal{L}1$

Entretanto, o modelo lógico não representa mudanças dessas entidades no tempo. Tampouco trata de entidades da UFO-B. Por exemplo, não é possível dizer que a *Emenda Constitucional nº 54, de 20 de setembro de 2009* altera a redação da *alínea c do inciso I do art. 12 da Constituição Federal* e nem que acrescenta o art. 95 ao *Ato das Disposições Constitucionais Transitórias*, assegurando o registro nos consulados de brasileiros nascidos no estrangeiro. Tampouco é possível dizer que a cor do cabelo de João era castanho em 1990 e grisalho em 2010.

Para tratar aspectos de mudanças temporais, realiza-se mais uma extensão ao modelo lógico $\mathcal{L}1$ e ao modelo de banco de dados, resultando nos modelos da Figura 3.16 e da Figura 3.17, respectivamente. Essas extensões visam representar *eventos*, entidades da UFO-B também conhecidas como *Perdurantes*.

Eventos são importantes para modelagem conceitual, para representação do conhecimento e para a cognição humana [104]. Por exemplo, Allen e March [118, p. 283, tradução nossa] afirmam que "o uso de diagramas Entidade-Relacionamento e correspondentes visões de usuário que dão destaque para alguma representação de eventos permite aos usuários formularem mais rapidamente consultas sem sacrificar precisão e e confi-

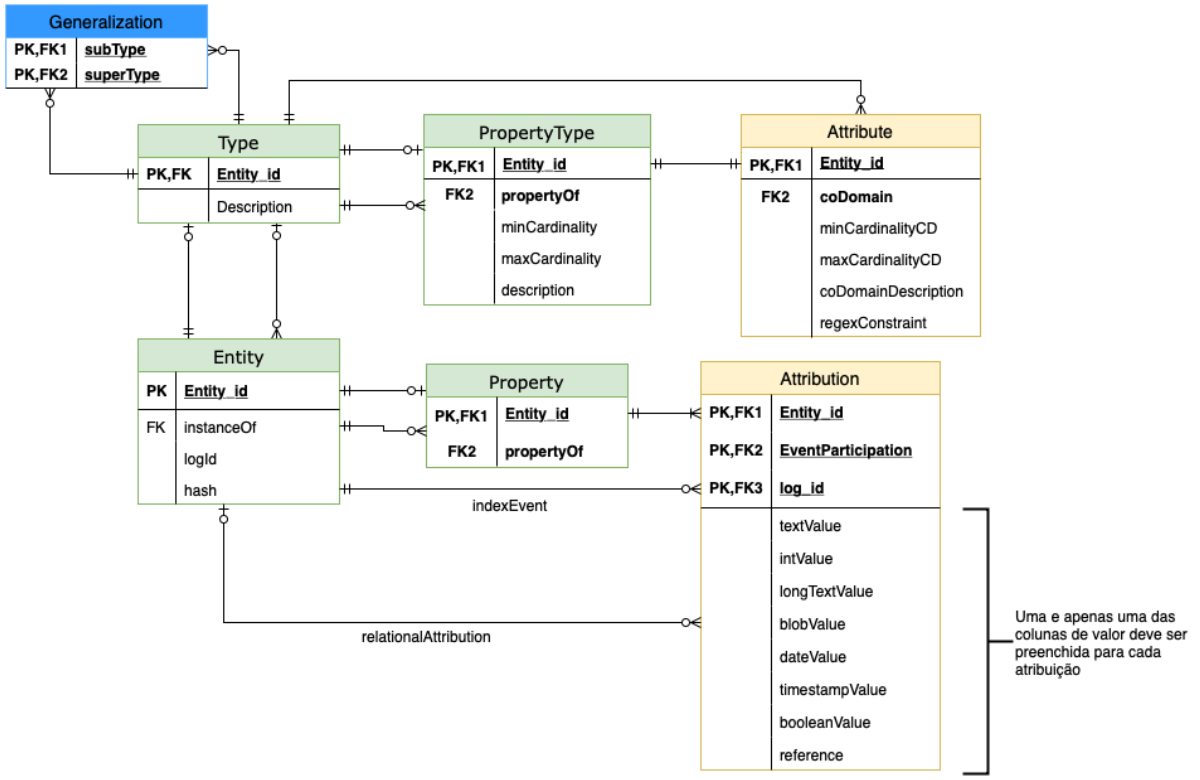


Figura 3.15: Extensão para a representação de especializações em banco de dados

ança[8]". Porém, eventos diferem de entidades endurantes em alguns aspectos, conforme discutido na Seção 2.7:

Apenas *eventos passados* são de interesse desta dissertação. Como consequência disso, eventos são imutáveis. Ou seja, uma vez ocorridos, suas propriedades não são mais alteradas [104].

Das categorias de Eventos da UFO-B apresentadas na Figura 2.15, abordam-se aqui apenas *Eventos Atômicos e Instantâneos*. Desta forma, por serem atômicos e instantâneos, os eventos apontam para um único ponto no tempo. Além disso, consoante ao exposto na Seção 2.7, um evento atômico e instantâneo depende diretamente e existencialmente de um objeto. Decorrente disso, apenas eventos relacionados a endurantes são modelados. Exemplos de Eventos Atômicos e Instantâneos:

- nascimento de *João*;
- entrada em vigor da *Lei Complementar nº 95, de 26 de fevereiro de 1998*;
- fim da partida entre *Novak Djokovic* e *Roger Federer* no torneio de Wimbledon de 2019;

[8] ... the use of E-R diagrams and corresponding user views that give prominence to some representation of events allow users to more quickly formulate queries without sacrificing accuracy or confidence

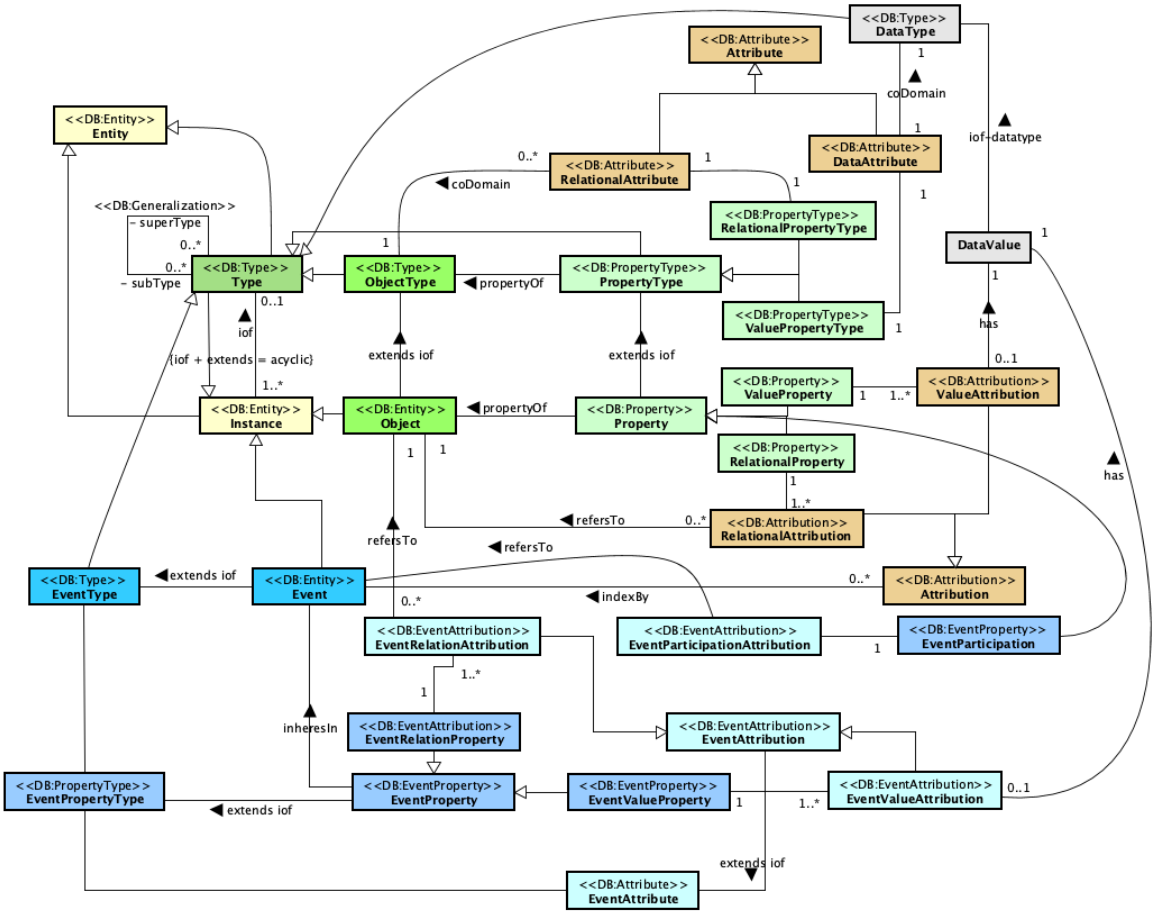


Figura 3.16: Modelo $\mathcal{L}1$ Evento

- momento em que *Eliud Kipchoge* cruza a linha de chegada e quebra o recorde mundial da Maratona no dia 16 de setembro de 2018 em Berlin;
- a declaração de independência do Brasil em 22 de abril de 1822.

Para permitir a representação de eventos, o modelo $\mathcal{L}1$ estendido da Figura 3.16 apresenta perdurantes em estrutura similar aos endurantes: *eventos* instanciam tipos de eventos, possuem propriedades relacionais e propriedades próprias, atribuições, valores e relacionam-se com *endurantes*.

A participação de objetos em eventos é representada em $\mathcal{L}1$ pela categoria $\mathcal{L}1$*:EventParticipation*. O relacionamento entre endurantes e eventos – como aqueles que indicam início ou fim de existência de endurantes – são assumidos como de natureza similar a propriedades relacionais de endurantes e, por isso, $\mathcal{L}1$*:EventParticipation* estende $\mathcal{L}1$*:Property*. Apesar da posição tomada, a discussão da natureza ontológica do relacionamento entre endurantes e eventos está fora do escopo deste trabalho.

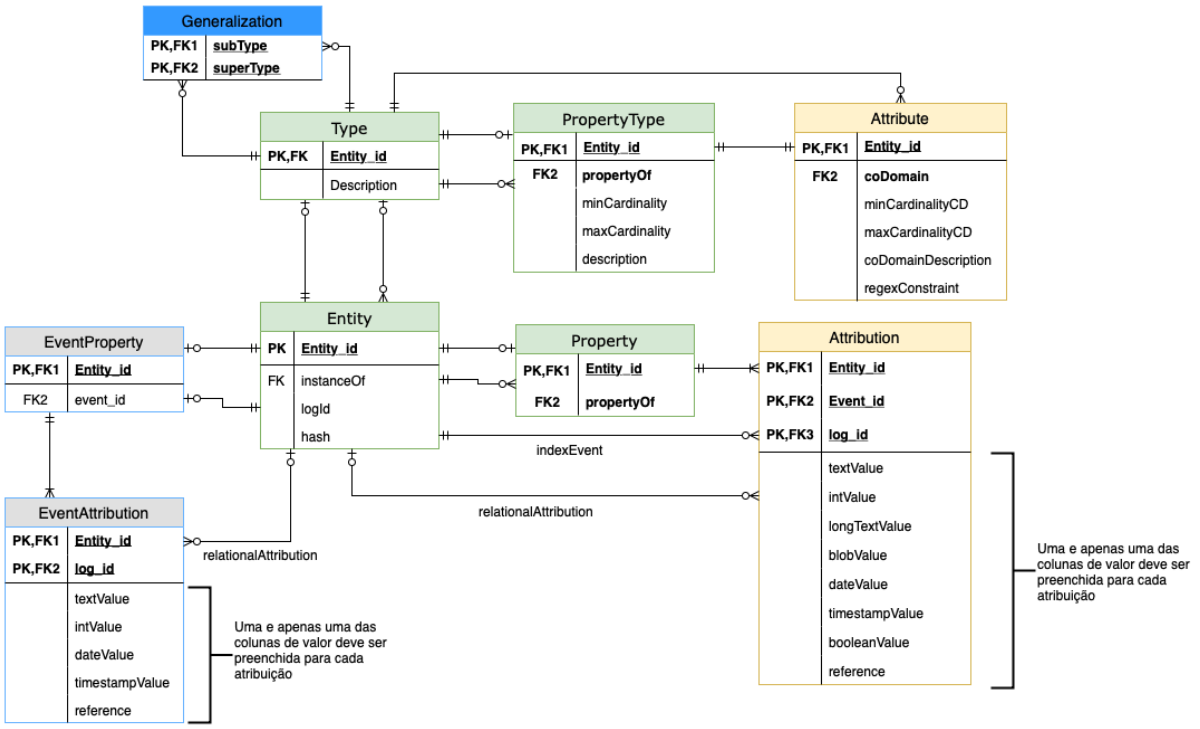


Figura 3.17: Extensão para indexação no tempo

$\mathcal{L}1$*:EventParticipation* difere de $\mathcal{L}1$*:EventRelationalProperty*. Por exemplo, a relação entre o endurante *Lei 8.666/1993* e o evento *Entrada em vigor da Lei 8.666/1993* representa uma participação de objetos em eventos. Em $\mathcal{L}1$*:EventParticipation*, o evento depende do endurante. Já se considerarmos termo como um objeto, o termo atribuído a um evento é um $\mathcal{L}1$*:EventRelationalProperty*. Neste último caso, o endurante depende do evento.

No nível de banco de dados, representa-se propriedades de eventos e atribuições de eventos em tabelas separadas pois eventos são imutáveis. Logo, as atribuições de endurantes são indexadas por um evento, enquanto atribuições de perdurantes não. Em outras palavras, uma propriedade de um endurante possui certo valor a partir da ocorrência do evento que o indexa. Por outro lado, uma propriedade de um perdurante sempre possuirá o mesmo valor na realidade.

O mapeamento de $\mathcal{L}1$ para o banco de dados é demonstrado na Figura 3.16. $\mathcal{L}1$*:EventTypes* são mapeados para a tabela *DB:Type* e suas instâncias para a tabela *DB:Entity*. As propriedades temporais de eventos bem como a participação de perdutantes em endurantes são mapeadas para a tabela *DB:EventProperty*.

Com a extensão apresentada nesta seção, é possível representar perdurantes, o relacionamento entre endurantes e perdurantes, bem como propriedades de eventos.

3.2.4 Eventos Epistemológicos

As extensões da Subseção 3.2.3 permitem representar as categorias presentes no quadrado aristotélico, no dodecágono ontológico e permitem indexar no tempo propriedades e valores de propriedades.

Apesar de eventos serem imutáveis e de cada propriedade ter apenas um valor ou estar relacionada a uma única instância em cada instante, é possível que valores ou referências à objetos sejam cadastrados com valores incorretos ou, no instante do registro, o que se sabe sobre valores e referências não reflitam a realidade.

Por isso, é importante diferenciar eventos de sistema, chamados aqui de registro (*Log*), de eventos de mundo. Ou, em outras palavras, diferenciar eventos ontológicos de eventos epistemológicos. Para isso, mais uma extensão é realizada ao modelo lógico $\mathcal{L}1$ e ao modelo de banco de dados: acrescenta-se a categoria $\mathcal{L}1$*:Log* e a tabela *DB:Log* ao modelo $\mathcal{L}1$ e de banco de dados, respectivamente. O mapeamento de $\mathcal{L}1$*:Log* é feito diretamente para a tabela *DB:Log*. O modelo $\mathcal{L}1$ completo é apresentado na Figura 3.18, já com todos os estereótipos de mapeamento para o modelo de banco de dados. Por sua vez, o modelo de banco de dados é exibido na Figura 3.19.

A categoria $\mathcal{L}1$*:Log* e a tabela *DB:Log* são responsáveis por manter informações de registro. Todas as atribuições de valores são indexadas, então, por um evento de mundo, ontológico, e por um evento de sistema, epistemológico. Dessa forma, é possível diferenciar correções de alterações de valores. A *correção* atribui um valor a uma propriedade sem alteração do evento de mundo que indexa a atribuição, ou seja, não houve nenhuma alteração de valor na realidade, apenas na representação em $\mathcal{L}1$ e na base de dados: a atribuição não mudou na realidade, apenas foi cadastrada incorretamente e, posteriormente, corrigida. Já uma *alteração* indica que um evento de mundo alterou uma atribuição de valor, ou seja, houve uma alteração na realidade e essa alteração foi representada na base de dados.

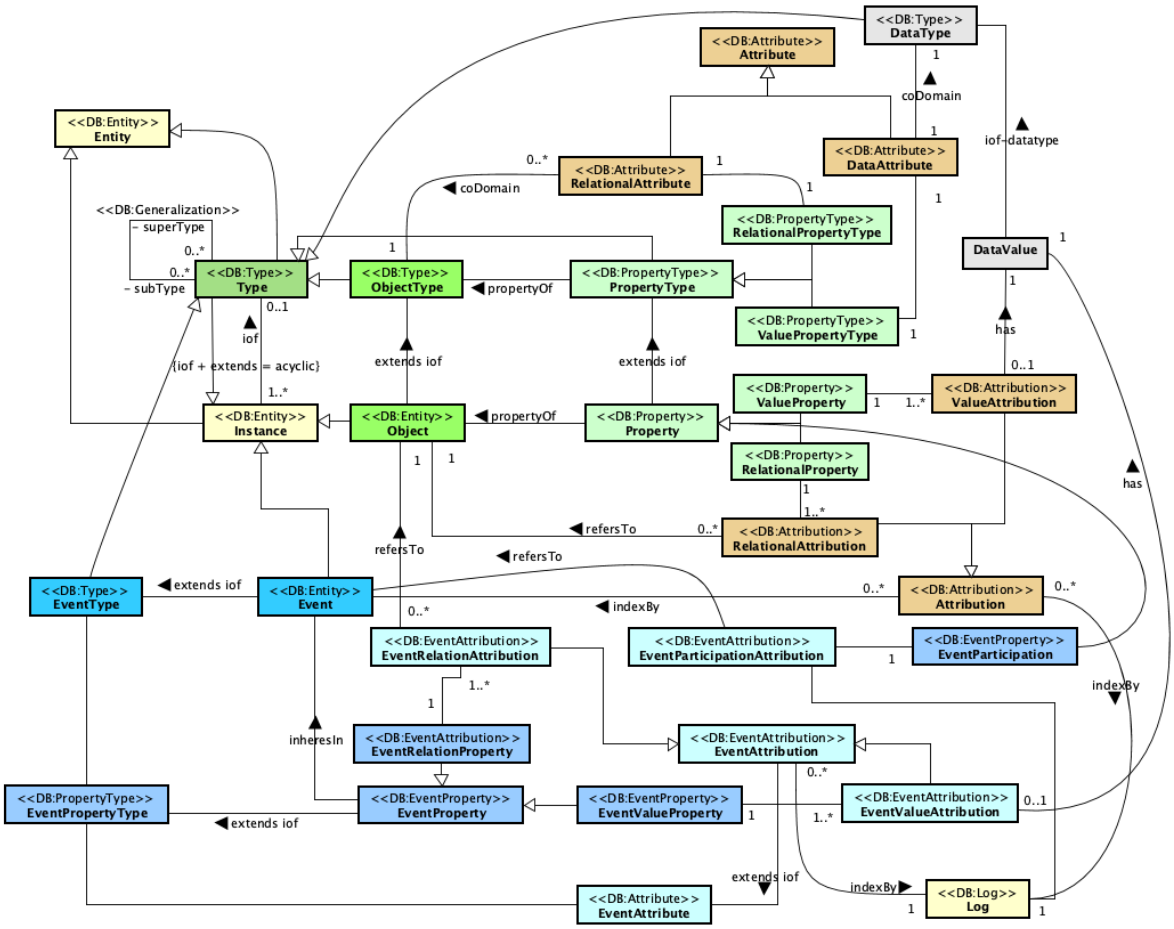


Figura 3.18: Modelo $\mathcal{L}1$ Completo

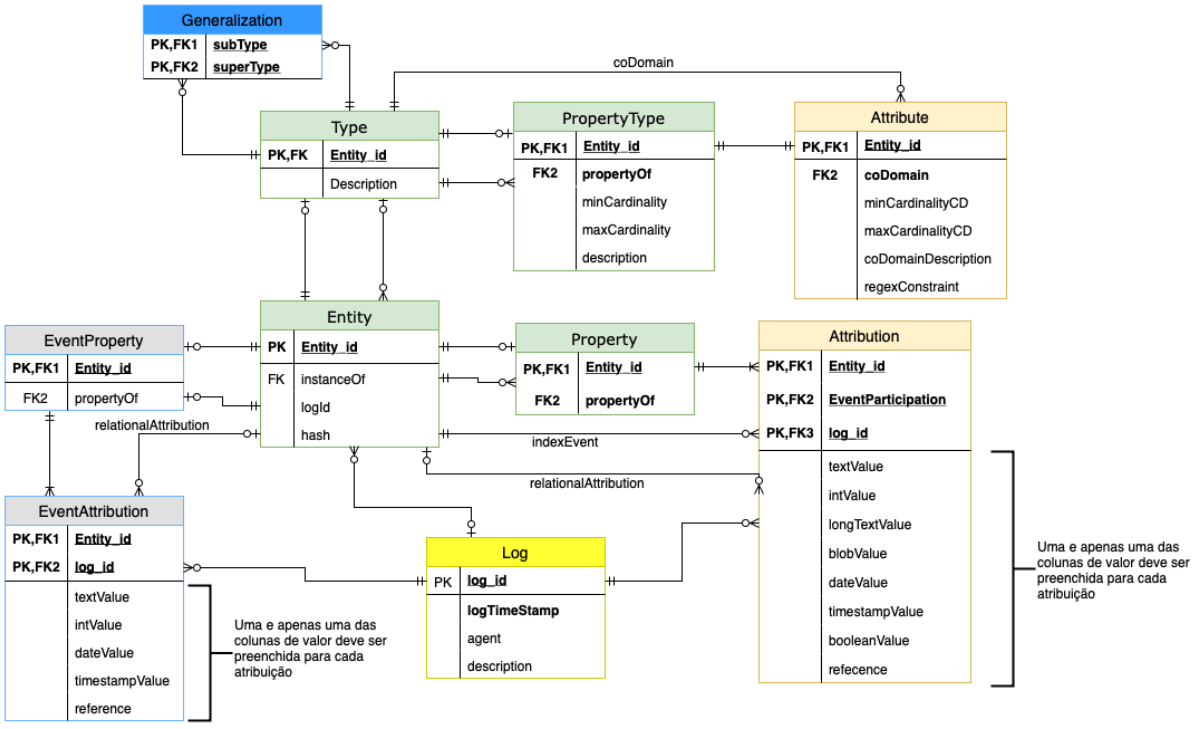


Figura 3.19: Extensão para diferenciar eventos do mundo e eventos de sistema

3.3 Restrições Sintáticas

Para mitigar problemas na utilização dos modelos como ciclos entre os relacionamentos de instanciação e especialização, ou de instanciação de categorias que não representam a natureza ontológica das entidades, implementações baseadas nesta pesquisa devem atender as restrições propostas nesta Seção, que trata de restrições sintáticas, e na Seção 4.7, que trata de restrições semânticas (especialmente de restrições UFO).

Algumas restrições sintáticas estão apenas especificadas, enquanto as restrições semânticas apresentadas estão necessariamente implementadas em SQL. A lista apresentada de restrições não é exaustiva.

Restrição 3.3.1. Categorias de domínio não podem ter o tipo que instanciam modificado, caso existam instâncias dessas categorias inseridas em banco.

Restrição 3.3.2. Caso uma instância seja cadastrada no modelo de banco de dados, a categoria que instancia só pode ser corrigida se todas as propriedades preenchidas para o tipo atual estejam também presentes na categoria alvo.

Restrição 3.3.3. O relacionamento de *instanciação* é irreflexível, antissimétrico e antitransitivo. Também só pode existir entre entidades de níveis adjacentes (Restrição MLT, Seção 2.9).

Restrição 3.3.4. O relacionamento de *especialização própria* é irreflexível, antissimétrico e transitivo. Também só ocorre entre indivíduos de mesmo nível. (Restrição MLT, Seção 2.9).

Restrição 3.3.5. O relacionamento de *instanciação* em conjunto com o relacionamento de *especialização própria* é irreflexível, antissimétrico e transitivo. Restrição necessária para evitar ciclos entre as os pares *DB:Entity*/*DB:Type* e $\mathcal{L}1$*:Entity*/$\mathcal{L}1$*:Type*, já que *DB:Entity* instancia *DB:Type*, mas *DB:Type* especializa *DB:Entity* e, do mesmo modo, $\mathcal{L}1$*:Entity* instancia $\mathcal{L}1$*:Type*, mas $\mathcal{L}1$*:Type* especializa $\mathcal{L}1$*:Type*.

Restrição 3.3.6. Propriedades devem ser propriedades de um único objeto.

A própria estrutura de banco de dados já garante algumas das restrições, como é o caso da irreflexibidade da instanciação. A camada lógica $\mathcal{L}1$ especifica as restrições acima, porém restrições adicionais podem ser incluídas nas implementações concretas.

Alguns exemplos de implementação são apresentados a seguir:

```
1  SELECT ID, COUNT(DISTINCT PROPERTY_OF) AS COUNT
2  FROM PROPERTY
3  GROUP BY ID
```

```
4     HAVING COUNT(DISTINCT PROPERTY_OF) > 1
5
```

Código 3.1: Verifica se há propriedades que são inerentes a mais de um objeto

```
1     SELECT A.ID AS ID_A, A.PROPERTY_OF AS A_BEARER, B.ID AS ID_B, B.
      PROPERTY_OF AS B_BEARER
2     FROM PROPERTY_TYPE A
3     INNER JOIN PROPERTY_TYPE B
4     ON A.ID = B.PROPERTY_OF
5     WHERE B.ID = A.PROPERTY_OF
6
```

Código 3.2: Verifica se caraterização é uma relação assimétrica

```
1         SELECT A.ID AS COD_A, A.INSTANCE_OF AS TYPE_A, B.ID AS COD_B, B.
      INSTANCE_OF AS TYPE_B
2         FROM ENTITY A
3         INNER JOIN ENTITY B
4         ON A.ID = B.INSTANCE_OF
5         WHERE B.ID = A.INSTANCE_OF
6
```

Código 3.3: Verificação da antissimetria do relacionamento de instanciação

```
1     SELECT ID, INSTANCE_OF
2     FROM ENTITY
3     WHERE ID = INSTANCE_OF
4
```

Código 3.4: Verificação da irreflexibidade do relacionamento de instanciação

```
1     SELECT A.ID AS COD_A, A.INSTANCE_OF AS TYPE_A, B.ID AS COD_B, B.
      INSTANCE_OF AS TYPE_B
2     FROM ENTITY A
3     INNER JOIN ENTITY B
4     ON A.ID = B.INSTANCE_OF
5     WHERE B.INSTANCE_OF = A.INSTANCE_OF
6
```

Código 3.5: Verificação da antitransitividade do relacionamento de instanciação

3.4 Considerações Finais

Este Capítulo apresentou a estrutura da solução para o problema de pesquisa. A solução é composta por um AOM em que o modelo de aplicação é inspirado no quadrado aristotélico e no dodecágono ontológico e o modelo de objetos baseado em UFO.

O modelo de aplicação compõe-se de duas camadas:

- uma camada lógica, nomeada como $\mathcal{L}1$, que apresenta a lógica para tradução do modelo de objetos em tuplas de banco de dados;
- uma camada física, de banco de dados, que apresenta a estrutura física do banco de dados que suportará o modelo de objetos.

O modelo de objetos, por sua vez, é composto pela UFO, categorias de domínio e indivíduos (objetos de domínio). Ele é apresentado em três níveis:

- nível meta-conceitual, que contém os Universais da UFO;
- nível conceitual, que contém os Particulares da UFO e as categorias de domínio;
- nível de indivíduos, que contém os Indivíduos do domínio.

Por fim, apresentam-se algumas restrições sintáticas dos modelos $\mathcal{L}1$ e de banco de dados.

Capítulo 4

Aplicação do Modelo de Objetos em Banco de Dados e Principais Resultados

Conforme apresentado na Seção 3.1, o modelo de objetos é representado no modelo de aplicação de modo que possa ser computacionalmente interpretado. O modelo de objetos possui entidades da UFO e de domínio divididos em três níveis: meta-conceitual, conceitual e de indivíduos. Todo o modelo de objetos deve ser persistido no banco de dados resultante da Figura 3.19. A inclusão do modelo de objetos em banco utiliza-se da lógica descrita no modelo lógico $\mathcal{L}1$ apresentado na Figura 3.18.

Esta Seção explica como as entidades da UFO e de domínio que compõem o modelo de objetos são mapeadas para $\mathcal{L}1$ e inseridas em banco de dados.

4.1 Inclusão de tipos de dados

Além da inclusão em banco de dados do modelo de objetos, deve-se também incluir a categoria *Datatype* e suas instâncias. *Datatype* está representada no modelo $\mathcal{L}1$ e no dodecágono ontológico. A Figura 3.11 apresenta o mapeamento da categoria do dodecágono *Datatype* para $\mathcal{L}1$*:Datatype*

A inclusão de qualquer entidade em banco de dados inicia-se pela tabela *DB:Entity*. *DB:Entity* contém apenas quatro colunas:

1. *id*, que é o identificador único da instância;
2. *instanceOf*, indicando o tipo instanciado pela entidade representada na tupla;
3. *logId*, que contém o código do registro que indexa os objetos do domínio. Esta coluna não é utilizada para categorias, apenas para objetos;

4. *hash*, coluna para gravar o resultado de uma função *hash* para permitir implementação de restrições de identidade, caso desejado.

Por ser uma categoria, *Datatype* deve ser também inserida na tabela *DB:Type*. A tabela *DB:Entity* representa entidades que instanciam tipos representados em *DB:Type*. Toda instância de *DB:Type*, por sua vez, é instância de *DB:Entity*. Como ciclos não são permitidos entre os relacionamentos de instanciação e especialização, existem entidades que não instanciam nenhum tipo no banco de dados. É o que ocorre com a categoria *Datatype*. A Figura 4.1 apresenta o estado do banco de dados após a inclusão da categoria: a tabela *DB:Entity* contém a identificação da entidade *Datatype* como chave primária e indica que a categoria não instancia nenhuma outra categoria. A tabela *DB:Type* referencia a chave primária de *DB:Entity* e apresenta uma descrição do tipo, no caso, o nome da categoria.

ENTITY			
ID	INSTANCE_OF	LOG_ID	HASH
1	null		

TYPE	
ID	DESCRIPTION
1	Datatype

Figura 4.1: Registro da entidade *Datatype* no banco de dados

Conforme mostra a Figura 3.18, a tabela mais específica em que instâncias de *Datatype* são armazenadas é *DB:Type*. A Figura 4.2 apresenta as tabelas *DB:Entity* e *DB:Type* preenchidas com instâncias de *Datatype*, cuja identificação é referenciada na coluna *instanceOf*.

Na inclusão tanto de *Datatypes* como de suas instâncias, apenas as duas tabelas mostradas são preenchidas. As demais permanecem sem tuplas inseridas.

ENTITY			
ID	INSTANCE_OF	LOG_ID	HASH
22	1		
23	1		
24	1		
25	1		

TYPE	
ID	DESCRIPTION
22	boolean
23	dateTime
24	integer
25	string

Figura 4.2: Registro das instâncias de *Datatype* no banco de dados

4.2 Inclusão da Taxonomia de Universais da UFO

Para permitir a inclusão de Particulares e, consequentemente, de objetos do domínio, necessita-se, primeiramente, representar os Universais da UFO. As categorias de Universais inseridas em banco são apresentadas na Figura 4.3.

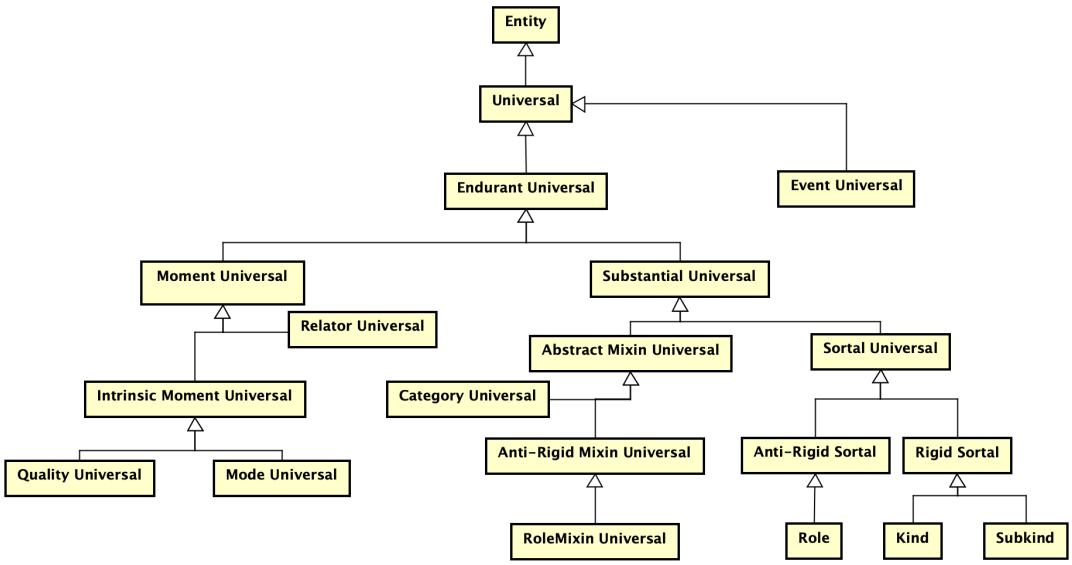


Figura 4.3: Fragmento de Universais da UFO

As categorias de Universais da UFO, nesta representação, não instanciam outras categorias. Apesar disso, o modelo permite que Universais sejam instâncias de categorias de ordem superior. Porém, optou-se por não representar mais uma ordem de entidades. Como consequência disso, a coluna *instanceOf* da tabela *DB:Instance* é nula para toda tupla que represente um Universal.

Então, de modo sucinto, as categorias de Universais são cadastradas no banco de dados em três tabelas:

1. *DB:Instance*: em que uma identificação para o Universal é indicada na coluna *id*, chave primária da tabela. A coluna *instanceOf* preenchida com valores nulos, já que se representou Universais como entidades de nível raiz, que não instanciam outras entidades;

2. *DB:Type*: em que a chave primária é também chave estrangeira e aponta para a chave primária de *DB:Instance*. O preenchimento desta tabela indica que um Universal também é um tipo;

3. *DB:Generalization*: em que as relações de especialização/generalização direta entre universais são representadas. Por exemplo, *IntrinsicMomentType*, *id* 10, especializa diretamente *MomentType*, *id* 8.

A Figura 4.4 representa o resultado da inclusão de Universais da UFO em banco de dados.

ENTITY

ID	INSTANCE_OF	LOG_ID	HASH
2	null		
4	null		
5	null		
6	null		
7	null		
8	null		
9	null		
10	null		
11	null		
12	null		
13	null		
14	null		
15	null		
16	null		
17	null		
18	null		
19	null		
20	null		
21	null		

TYPE

ID	DESCRIPTION
2	Entity
4	EndurantType
5	AbstractMixinType
6	AtomicEventType
7	CategoryType
8	MomentType
9	Universal
10	IntrinsicMomentType
11	Kind
12	MixinType
13	QuaIndividualType
14	PerdurantType
15	QualityType
16	RelatorType
17	SubstantialType
18	RoleMixinType
19	SortalType
20	RoleType
21	Subkind

GENERALIZATION

SUB_TYPE	SUPER_TYPE
9	2
8	4
17	4
7	5
12	5
18	5
10	8
16	8
4	9
14	9
13	10
15	10
6	14
5	17
19	17
11	19
20	19
21	19

SUB_TYPE_DESC	SUPER_TYPE_DESC
Universal	Entity
MomentType	EndurantType
SubstantialType	EndurantType
CategoryType	AbstractMixinType
MixinType	AbstractMixinType
RoleMixinType	AbstractMixinType
IntrinsicMomentType	MomentType
RelatorType	MomentType
EndurantType	Universal
PerdurantType	Universal
QuaIndividualType	IntrinsicMomentType
QualityType	IntrinsicMomentType
AtomicEventType	PerdurantType
AbstractMixinType	SubstantialType
SortalType	SubstantialType
Kind	SortalType
RoleType	SortalType
Subkind	SortalType

Figura 4.4: Resultado da inclusão dos Universais da UFO em banco de dados

4.3 Inclusão da Taxonomia de Particulares da UFO

As categorias da UFO são mapeadas para o modelo $\mathcal{L}1$ conforme Figura 4.5. Instâncias dos Universais apresentados são instâncias da categoria de $\mathcal{L}1$ referenciada no estereótipo de cada categoria. Da mesma forma, o mapeamento de $\mathcal{L}1$ para banco de dados discutido na Seção 3.2 e sumarizado na Figura 3.18 indica que uma instância de determinada categoria de $\mathcal{L}1$ é tupla da tabela referenciada no estereótipo da categoria de $\mathcal{L}1$.

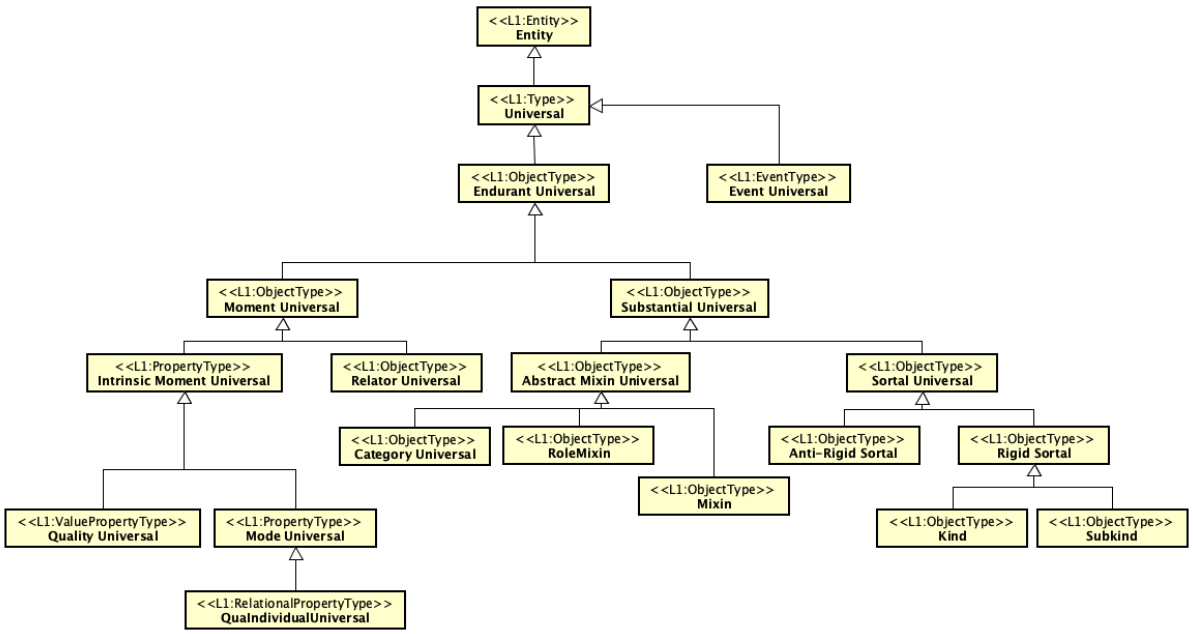


Figura 4.5: Mapeamento dos Universais da UFO para $\mathcal{L}1$

Desta forma, com o auxílio da Figura 3.18, da Figura 3.19 e da Figura 4.5, é possível mostrar a cadeia de transformações que levam a inclusão de Particulares (instâncias de de Universais) em Banco de Dados.

Por exemplo, o Particular *Substantial* é instância de *Substantial Universal* e *Quality* é instância de *Quality Universal*. Como *Substantial Universal* é mapeado para $\mathcal{L}1$*:ObjectType* e, por sua vez, $\mathcal{L}1$*:ObjectType* é mapeado para a tabela *DB:Type*, então instâncias de *Substantial Universal* são inseridas na tabela *DB:Type*. Logo, o particular *Quality*, instância de *Quality Universal*, é inserido em *DB:Type*. Do mesmo modo, *Quality Universal* é mapeado para $\mathcal{L}1$*:ValuePropertyType*, que por sua vez é mapeado para *DB:PropertyType*. Então, instâncias de *Quality Universal* são inseridas em *DB:PropertyType*. Logo, o particular *Quality*, como instância de *Quality Universal*, é inserido em *DB:PropertyType*.

A Figura 4.6 mostra o resultado da inclusão dos particulares da UFO nas tabelas *DB:Entity* e *DB:Type*. A Figura 4.7 apresenta as relações de especialização entre os Particulares inseridos. A Figura 4.8 aponta as três categorias de particulares que são propriedades. E, por fim, a Figura 4.9 contém os atributos que ligam as propriedades ao seu contradomínio.

ENTITY

ID	INSTANCE_OF	LOG_ID	HASH
26	9		
29	4		
30	8		
32	10		
41	15		
43	14		
48	19		
57	13		
59	16		
61	17		

INSTANCE_DESC	INSTANCE_OF_DESC
Particular	Universal
Endurant	EndurantType
Moment	MomentType
IntrinsicMoment	IntrinsicMomentType
Quality	QualityType
Perdurant	PerdurantType
FuncionalComplex	SortalType
QuaIndividual	QuaIndividualType
Relator	RelatorType
Substantial	SubstantialType

TYPE

ID	DESCRIPTION
26	Particular
29	Endurant
30	Moment
32	IntrinsicMoment
41	Quality
43	Perdurant
48	FuncionalComplex
57	QuaIndividual
59	Relator
61	Substantial

Figura 4.6: Resultado da inclusão dos particulares da UFO em banco de dados

GENERALIZATION			
SUB_TYPE	SUPER_TYPE	SUB_TYPE_DESC	SUPER_TYPE_DESC
26	3	Particular	Entity
29	26	Endurant	Particular
30	29	Moment	Endurant
32	30	IntrinsicMoment	Moment
41	32	Quality	IntrinsicMoment
43	26	Perdurant	Particular
48	61	FuncionalComplex	Substantial
57	32	QuaIndividual	IntrinsicMoment
59	30	Relator	Moment
61	29	Substantial	Endurant

Figura 4.7: Especialização: Particulares da UFO

PROPERTY_TYPE				
ID	PROPERTY_OF	MIN_CARDINALITY	MAX_CARDINALITY	DESCRIPTION
32	26	0	99999	Instrinsic Moment
41	26	0	99999	Quality D
57	26	0	99999	QuaIndividual

Figura 4.8: Particulares inseridos em DB:PropertyType

ATTRIBUTE					
ID	CODOMAIN	MIN_CARD	MAX_CARD	DESCRIPTION	CODOMAIN_CONSTRAINT
32	26	0	99999	Instrinsic Moment of	
41	26	0	99999	Quality of	
57	26	0	99999	QuaIndividual in	

Figura 4.9: Tabela Attribute ligando os propertyTypes ao seu contradomínio

4.4 Inclusão de Categorias do Domínio

A inclusão das categorias de domínio é similar a inclusão de Particulares da UFO, pois também são instâncias de Universais. A Figura 4.10 apresenta um exemplo de categorias relacionadas ao domínio casamento. Apesar de ser mais comum encontrar modelagens enfatizando os papeis (*Roles*), este exemplo modela os *qua indivíduos*, pois eles são persistidos em banco. Com uma modelagem com *Roles*, é possível derivar os *qua indivíduos* necessários para representação no modelo de aplicação. A Figura 4.10 apresenta estereótipos com notação OntoUML, ou seja, uma categoria com estereótipo instancia o Universal da UFO citado no estereótipo.

Como apresenta a Figura 3.1, as categorias de domínio possuem relacionamento tanto com os Particulares quanto com os Universais da UFO. A Figura 4.11 explicita a cadeia

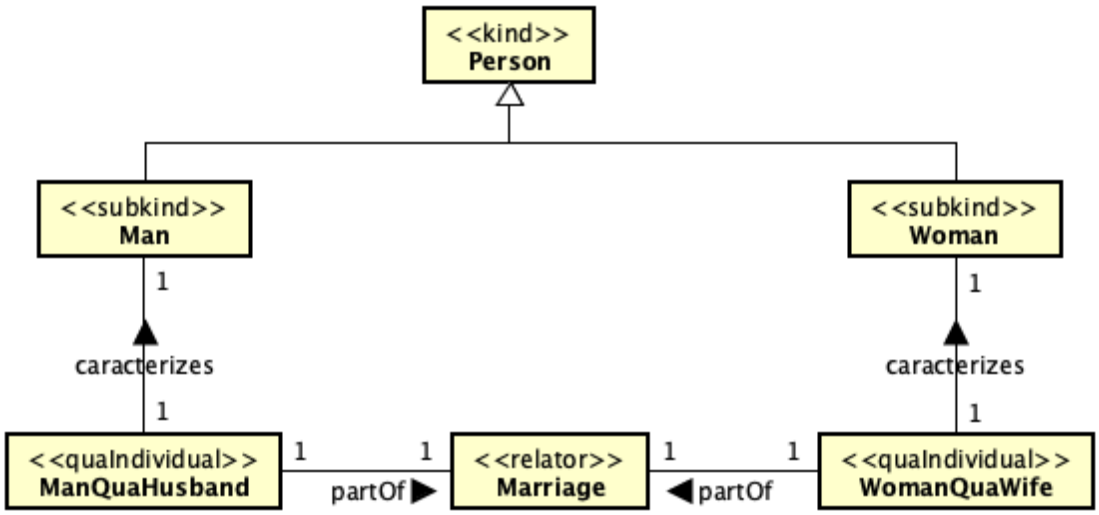


Figura 4.10: Exemplo de categorias do domínio casamento. Universais da UFO representados como estereótipos

de Particulares UFO especializados pelas categorias do exemplo. A Figura 4.12 mostra a cadeia de Universais e esclarece aqueles instanciados diretamente pelas categorias do domínio exemplificado. A Figura 4.13 apresenta os relacionamentos de instanciação e especialização em um diagrama unificado.

As Figuras também apresentam o mapeamento das categorias de domínio para $\mathcal{L}1$. O mapeamento para $\mathcal{L}1$ é útil para a inclusão dos objetos do domínio, instâncias das categorias apresentadas.

Outro detalhe importante desse exemplo é a que ele mostra como *relators* são representados no nível de categorias. Uma categoria que instancie *Relator Universal*, por exemplo, *Marriage*, conecta pelo menos duas categorias de *qua indivíduos*, por exemplo, *ManQuaHusband* e *WomanQuaWife*.

A Figura 4.14 apresenta as categorias do exemplo casamento inseridas nas tabelas *DB:Type* e *DB:Entity*. Ao lado da tabela *DB:Entity* encontra-se uma descrição da categoria e do universal que instancia. A descrição dos *qua indivíduos* contém o nome dos *Roles* (papeis) que exercem. Assim, a categoria de *qua indivíduos ManQuaHusband* está descrita apenas como *Husband*. Do mesmo modo, a categoria *WomanQuaWife* é descrita apenas como *Wife*. A Figura também permite visualizar os relacionamentos de instanciação entre as categorias de domínio e os universais da UFO.

A Figura 4.15 apresenta os relacionamentos de especialização direta entre as categorias

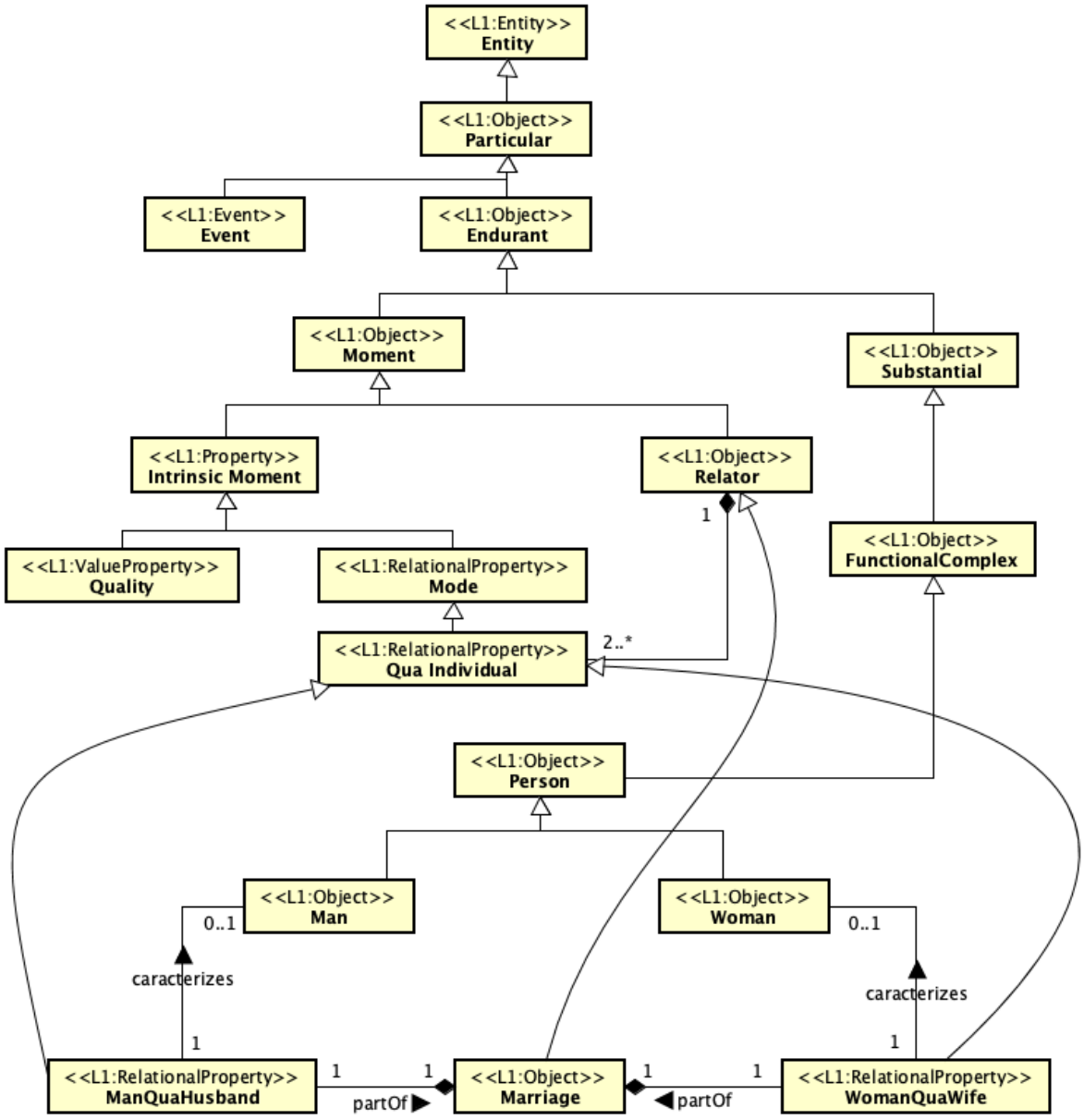


Figura 4.11: Exemplo de domínio, especialização de particulares da UFO e mapeamento para $\mathcal{L}1$

de domínio e os particulares da UFO, como, por exemplo, *woman* e *man*, especializações de *person*, e *marriage*, especialização de *relator*. A Figura 4.16 mostra que *ManQuaHusband* é propriedade de *Man* e *WomanQuaWife* é propriedade de *Wife*. A Figura 4.17 revela que o contradomínio dos *qua indivíduos ManQuaHusband* e *WomanQuaWife* é o *relator Marriage*.

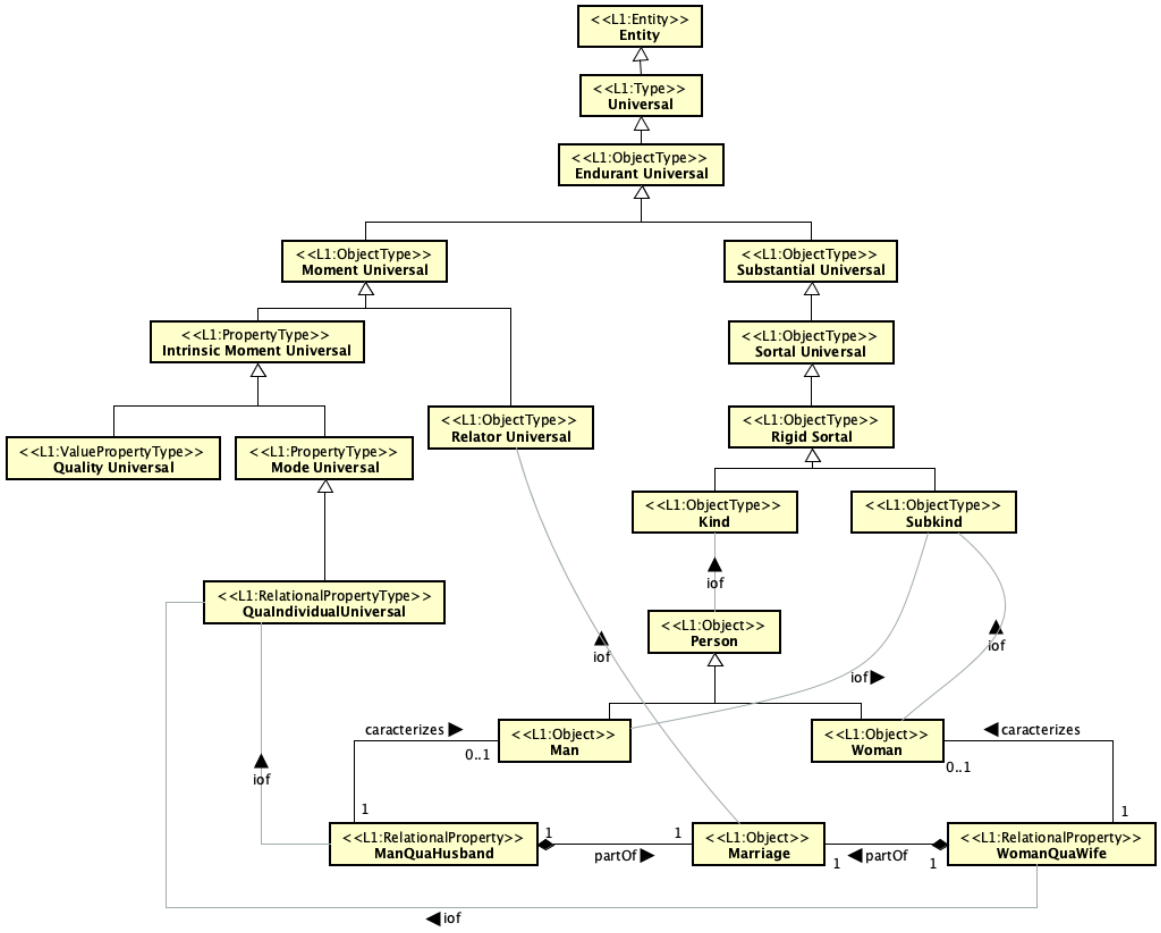


Figura 4.12: Exemplo de domínio, universais que instanciam e o mapeamento para $\mathcal{L}1$

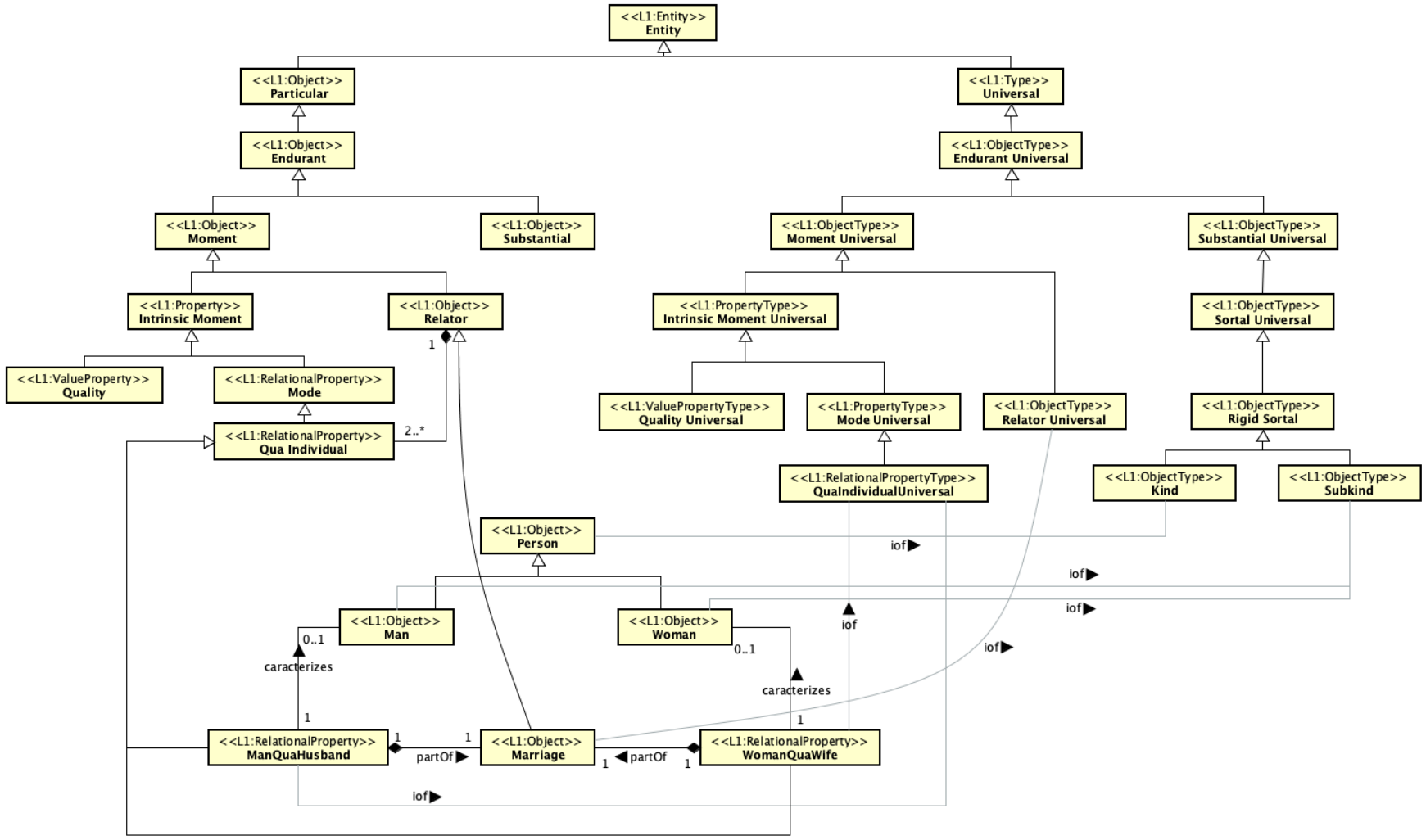


Figura 4.13: Mapeamento completo do exemplo de domínio para $\mathcal{L}1$

ENTITY			
ID	INSTANCE_OF	LOG_ID	HASH
63	11		
67	21		
68	21		
69	13		
70	16		
71	13		

INSTANCE_DESC	INSTANCE_OF_DESC
Person	Kind
Man	Subkind
Woman	Subkind
Husband	QuaIndividualType
Marriage	RelatorType
Wife	QuaIndividualType

TYPE	
ID	DESCRIPTION
63	Person
67	Man
68	Woman
69	Husband
70	Marriage
71	Wife

Figura 4.14: Inclusão em banco de categorias de domínio: tabelas *DB:Type* e *DB:Entity*

GENERALIZATION	
SUB_TYPE	SUPER_TYPE
63	48
67	63
68	63
69	57
70	59
71	57

SUB_TYPE_DESC	SUPER_TYPE_DESC
Person	FuncionalComplex
Man	Person
Woman	Person
Husband	QuaIndividual
Marriage	Relator
Wife	QuaIndividual

Figura 4.15: Inclusão em banco de categorias de domínio: tabela *DB:Generalization*

PROPERTY_TYPE				
ID	PROPERTY_OF	MIN_CARDINALITY	MAX_CARDINALITY	DESCRIPTION
69	67	1	1	Husband
71	68	1	1	Wife

Figura 4.16: Inclusão em banco de categorias de domínio: tabela *DB:PropertyType*

ATTRIBUTE					
ID	CODOMAIN	MIN_CARD	MAX_CARD	DESCRIPTION	CODOMAIN_CONSTRAINT
69	70	0	1	Husband in	
71	70	0	1	Wife in	

Figura 4.17: Inclusão em banco de categorias de domínio: tabela *DB:Attribute*

4.5 Subsistema de Nomes

Este trabalho utiliza um subsistema de nomes para identificar objetos. As categorias do subsistema de nomes são apresentadas na Figura 4.18. O próprio subsistema de nomes já se utiliza da estrutura oferecida por este trabalho. Ou seja, esse subsistema é um conjunto de categorias e relacionamentos modelados com a UFO, mapeado para $\mathcal{L}1$ e para tabelas de banco de dados e, consequentemente, inserido como tuplas em banco tal como suas instâncias.

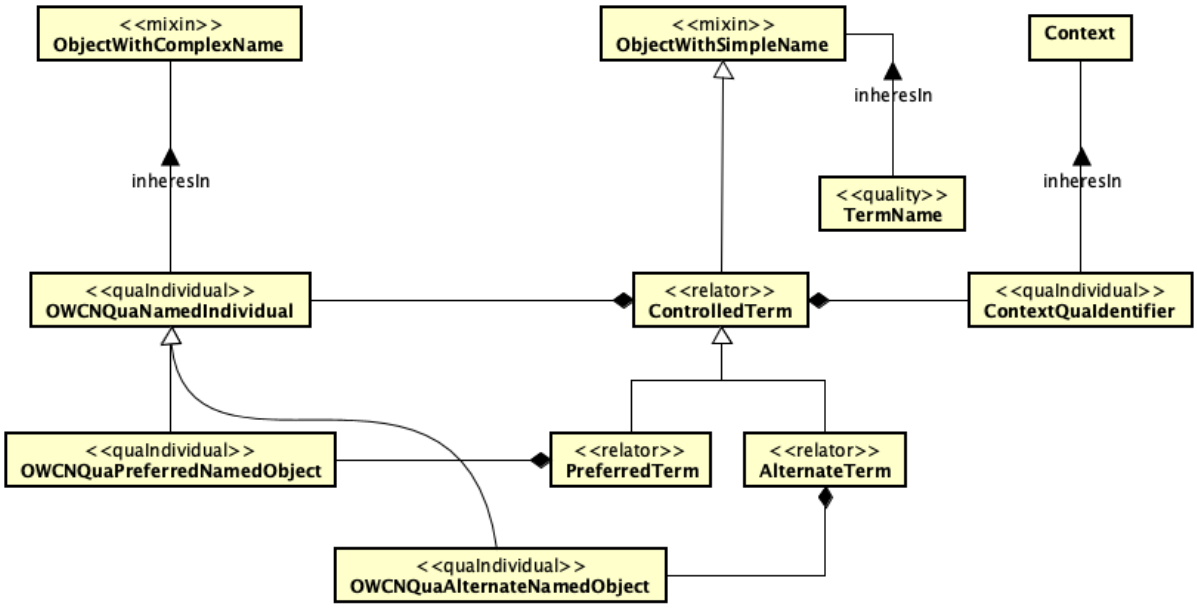


Figura 4.18: Categorias do subsistema de nomes

O pré-cadastro de categorias de nomes oferece flexibilidade para nomear objetos. Essa abordagem separa o subsistema de nomes do subsistema de categorias. Desta forma, para uma categoria ter um nome complexo, com opções de nomes alternativos e preferidos, basta que ela especialize o *mixin ObjectWithComplexName*. Para categorias com nome simples, ou seja, com uma única opção de nome, basta especializar o *mixin ObjectWithSimpleName*.

O subsistema de nomes considera a categoria *termo controlado* como um *relator* que media um *Objeto com Nome Complexo* e um *Contexto*. O tenista sérvio *Novak Djokovic*, por exemplo, é conhecido por diferentes termos em contextos diferentes: *Novak Djokovic*, para a comunidade internacional; *Novak Đoković*, em seu país natal; *Nole*, entre sua família e amigos mais próximos; em outros contextos ainda é conhecido como *Djoker*, *The joker* ou *Djoko*. O contexto, por sua vez, é um objeto complexo com vários participantes. Por questões de simplicidade, optamos por omitir o contexto na representação em banco

e também por não detalhar suas características ontológicas. Desta forma, os objetos com nome complexos são nomeados por um *termo preferido* e por zero ou mais *termos alternativos*, apenas. No exemplo de *Djokovic*, "Novak Djokovic" pode ser escolhido como termo preferido enquanto os demais nomes como termos alternativos.

As Figuras 4.19 à 4.22 apresentam as tuplas em banco de dados correspondentes às categorias de nomes.

ENTITY

ID	INSTANCE_OF	LOG_ID	HASH
27	16		
33	13		
36	12		
37	12		
38	16		
40	12		
41	15		
50	15		
51	13		
52	12		
53	11		
55	12		
56	16		
58	13		
60	15		
62	15		

INSTANCE_DESC	INSTANCE_OF_DESC
AlternateTerm	RelatorType
AlternateNamedObject	QuaIndividualType
SystemCategory	MixinType
ObjectWithSimpleName	MixinType
ControlledTerm	RelatorType
ObjectWithComplexName	MixinType
Quality	QualityType
Name	QualityType
NamedIndividual	QuaIndividualType
NamedObject	MixinType
ObjectWithInferredName	Kind
PreferredTerm	MixinType
ObjectWithUniquenessConstraint	RelatorType
PreferredNamedObject	QuaIndividualType
StringQuality	QualityType
TermName	QualityType

TYPE

ID	DESCRIPTION
27	AlternateTerm
33	AlternateNamedObject
36	SystemCategory
37	ObjectWithSimpleName
38	ControlledTerm
40	ObjectWithComplexName
41	Quality
50	Name
51	NamedIndividual
52	NamedObject
53	ObjectWithInferredName
56	PreferredTerm
55	ObjectWithUniquenessConstraint
58	PreferredNamedObject
60	StringQuality
62	TermName

Figura 4.19: Categorias do Domínio de Nomes como instâncias das tabelas *DB:Entity* e *DB:Type*

GENERALIZATION

SUB_TYPE	SUPER_TYPE
27	38
33	51
36	61
37	52
38	37
38	59
40	52
41	32
50	60
51	57
52	36
53	52
56	38
56	55
58	51
60	41
62	50

SUB_TYPE_DESC	SUPER_TYPE_DESC
AlternateTerm	ControlledTerm
AlternateNamedObject	NamedIndividual
SystemCategory	Substantial
ObjectWithSimpleName	NamedObject
ControlledTerm	ObjectWithSimpleName
ControlledTerm	Relator
ObjectWithComplexName	NamedObject
Quality	IntrinsicMoment
Name	StringQuality
NamedIndividual	QuaIndividual
NamedObject	SystemCategory
ObjectWithInferredName	NamedObject
PreferredTerm	ControlledTerm
PreferredTerm	ObjectWithUniquenessConstraint
PreferredNamedObject	NamedIndividual
StringQuality	Quality
TermName	Name

Figura 4.20: Tabela *DB:Generalization* mostrando as especializações entre categorias do domínio de nomes

PROPERTY_TYPE				
ID	PROPERTY_OF	MIN_CARDINALITY	MAX_CARDINALITY	DESCRIPTION
33	40	1	1	Alternate Term of
41	26	0	99999	Quality
50	26	0	99999	Name
51	40	0	99999	Named Individual
58	40	1	1	Preferred Term of
60	26	0	99999	String Quality
62	37	1	1	Term Name

Figura 4.21: Tabela *DB:PropertyType* com as categorias de propriedades do Domínio de Nomes

ATTRIBUTE					
ID	CODOMAIN	MIN_CARD	MAX_CARD	DESCRIPTION	CODOMAIN_CONSTRAINT
33	27	0	99999	Alternate Term	
41	26	0	99999	Quality CD	
50	25	0	99999	Name CD	
51	38	0	99999	Named Individual in	
58	56	1	1	Preferred Term	
60	25	0	99999	String Quality CD	
62	25	1	1	Name	

Figura 4.22: Tabela *DB:Attribute* ligando as categorias de propriedades

4.6 Inclusão de Objetos de Domínio

Normalmente, modela-se software para permitir a representação de indivíduos existentes na realidade. Neste trabalho, as categorias da UFO foram utilizadas para produção de categorias de domínio e relacionamentos ontologicamente bem fundamentados. As categorias de domínio, por sua vez, são utilizadas para permitir a representação em software de indivíduos de um domínio.

Logo, as inclusões em banco de dados das categorias da UFO e das categorias de domínio – em outras palavras, dos níveis meta-conceitual e conceitual – são necessárias para permitir a interpretação computacional do modelo de objetos de modo a possibilitar a representação de objetos de domínio.

Esta Seção traz exemplos simples de como instâncias das categorias apresentadas na Figura 4.10 são inseridas em banco de dados. A primeira inclusão, de acordo com a Figura 4.23, é de um indivíduo, instância de homem, que possui um nome: *Liev Nikoláievich Tolstói*. Posteriormente, já com o resultado da inclusão de outro indivíduo com nome, *Sophia Behrs*, inclui-se o relator *casamento de Tolstoi e Sophia* e os *qua indivíduos* intrínsecos à *Tolstói* e à *Sophia*, conforme objetos apresentados na Figura 4.24.

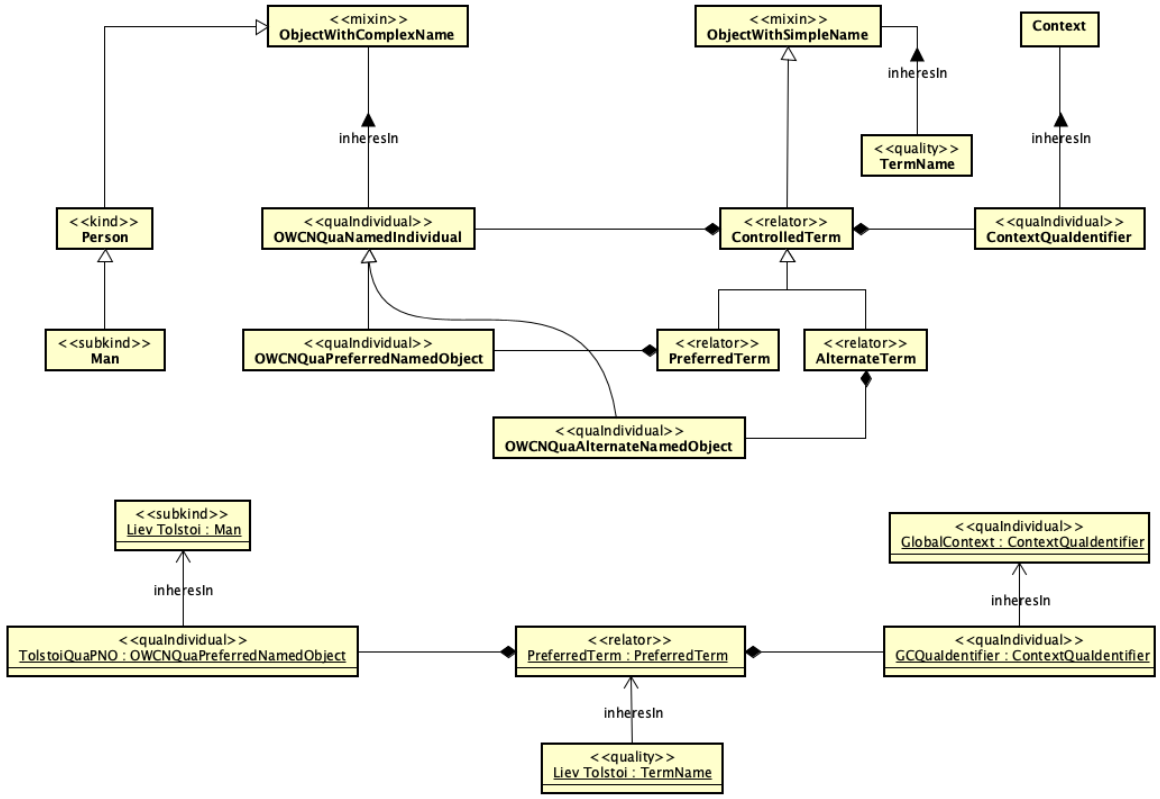


Figura 4.23: Exemplo de objetos de domínio: indivíduo Tolstoi e seu nome

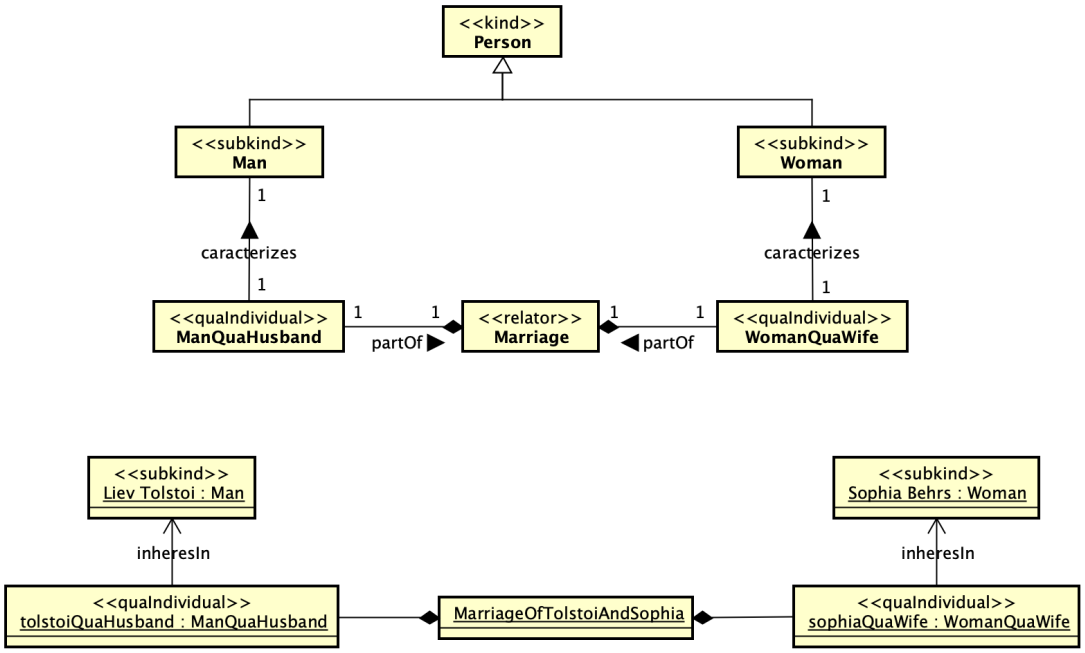


Figura 4.24: Exemplo de objetos de domínio: Casamento de Leon Tolstoi e Sophia Behrs

Analisando as categorias de domínio, seus relacionamentos com UFO apresentados na Figura 4.23 e na Figura 4.24 e sabendo-se que essas categorias já estão persistidas em banco de dados, pode-se, então, cadastrar os objetos de domínio.

A Figura 4.23 possui os seguintes objetos de domínio:

- um indivíduo α instância de *Man*;
- um qua indivíduo β inerente à α e instância de *ObjectWithComplexNameQuaNamedIndividual*;
- um relator γ, instância de *Preferred Term* que media o indivíduo α e o contexto ϵ em que esse nome é usado;
- um nome de termo δ, instância de *Term name*, inerente à γ e que contém o valor *"Liev Nikoláievich Tolstói"*;
- um contexto ϵ e consequente qua indivíduo não representados em banco para simplificar o exemplo.

Analisando as Figuras 4.23, 4.11, 3.18 e 3.19, tem-se que a inclusão dos objetos de domínio são feitos da seguinte forma:

- a categoria *Man* é mapeada para $\mathcal{L}1$*:Object* (Figura 4.11). Por sua vez, $\mathcal{L}1$*:Object* é mapeada para *DB:Entity*. Logo, α como instância de *Man*, é cadastrado apenas em *DB:Entity*;
- a categoria *ObjectWithComplexNameQuaNamedIndividual* é mapeada para $\mathcal{L}1$*:RelationalProperty*, pois é especialização da categoria UFO *QuaIndividual* (não confundir com o estereótipo *quaIndividual* da 4.23, que indica o universal instanciado). Logo, β é mapeada para *DB:Property* como propriedade de α;
- γ é inserido em *DB:Object*, já que *Preferred Term* é categoria de relators mapeada para $\mathcal{L}1$*:Object*, que por sua vez é mapeada para *DB:Object*;
- δ é instância de *Term Name*. Logo, por ser especialização de *quality*, é mapeada para $\mathcal{L}1$*:ValueProperty*, que por sua vez é mapeada para *DB:Property*. A atribuição de δ para o valor *"Liev Nikoláievich Tolstói"*, instância de *string*, é feita na tabela *DB:Attribution*.

A Figura 4.25 mostra a inclusão dos indivíduos α, β, γ e δ em na tabela *DB:Entity*. Ao lado do conteúdo da tabela, há uma legenda indicando a categoria de domínio instanciada por cada indivíduo. A Figura 4.26 apresenta as propriedades β e δ como propriedades de α e γ, respectivamente. Por fim, a Figura 4.27 relaciona as propriedades β e δ a um valor. A propriedade β, por ser um qua indivíduo, referencia o identificador do *relator*

termo controlado (o indivíduo γ). A propriedade δ, por sua vez, referencia o valor texto *"Liev Nikoláievich Tolstói"*.

	ENTITY				
	ID	INSTANCE_OF	LOG_ID	HASH	INSTANCE_OF_DESC
α	74	67	1		Man
β	76	58	1		PreferredNamedObject
γ	79	56	1		PreferredTerm
δ	80	62	1		TermName

Figura 4.25: Inclusão de objetos de domínio em *DB:Entity*

	PROPERTY		
	ENTITY_ID	PROPERTY_OF	
β	76	74	α
δ	80	79	γ

Figura 4.26: Inclusão de propriedades de domínio em *DB:Property*

	ATTRIBUTION								
	ENTITY_ID	EVENT_PARTICIPATION	LOG_ID	TEXT_VALUE	LONG_TEXT_VALUE	INT_VALUE	DATE_VALUE	BOOLEAN_VALUE	REFERENCE
β	76	81	1						79 γ
δ	80	81	1	Liev Nikoláievich Tolstói					

Figura 4.27: Atribuição de valor à propriedade relacional β e à propriedade valorada δ

As Figuras 4.25 e 4.27 apresentam ainda colunas de eventos, no caso a coluna *log* de *DB:Entity* além das colunas *log* e *eventParticipation* de *DB:Attribution*. O tratamento de eventos em banco de dados é explicado na Subseção 4.6.1 .

4.6.1 Eventos: inclusão e relacionamentos

Consoante o apresentado na Subseção 3.2.3 e na Subseção 3.2.4, o modelo lógico $\mathcal{L}1$ e o modelo físico de banco de dados possuem capacidade para registrar eventos atômicos e seus relacionamentos com endurantes. A atribuição de valores de propriedades de endurantes, conforme mostra a Figura 4.27, é indexada por um evento de mundo, identificado pela coluna *eventParticipation*, e por um evento de sistema, identificado pela coluna *log*.

A Figura 4.28 apresenta dois eventos de mundo: *Tolstoi's Birth* (nascimento de Tolstoi, id: 77) e *Tolstoi's Name Creation Event* (evento de criação do nome de Tolstoi, id: 81).

Este último indexa duas propriedades: a atribuição do termo preferido ao indivíduo Tolstoi e a atribuição do nome do termo ao termo preferido, ou seja, a cadeia de caracteres "Liev Nikoláievich Tolstói" é atribuída à propriedade nome do termo e o termo é atribuído ao indivíduo Tolstoi durante o mesmo evento: a criação do nome de Tolstoi.

	ENTITY				
	ID	INSTANCE_OF	LOG_ID	HASH	INSTANCE_OF_DESC
Liev Tolstoi	74	67	1		Man
Tolstoi's Participation in his birth	75	66	1		HumanBirthParticipation
Tolstoi qua Preferred Named Object	76	58	1		PreferredNamedObject
Tolstoi's Birth	77	65	1		BirthEvent
Tolstoi's Birth Date	78	64	1		BirthDate
Tolstoi's preferred term	79	56	1		PreferredTerm
Tolstoi's preferred term name	80	62	1		TermName
Tolstoi's Name Creation Event	81	28	1		CreationEvent
Toltoi's Name Participation in its creation	82	47	1		EndurantCreation
Toistoi's Name Creation Event Date	83	45	1		EventDate

Figura 4.28: Eventos relacionados à Tolstoi em *DB:Entity*

Eventos, diferentemente de Endurantes, são imutáveis (Seção 2.7). Da mesma forma, a participação de objetos em eventos é imutável: uma vez ocorrida, não pode ser alterada. Portanto, nem propriedades de eventos nem a participação de objetos em eventos são indexadas por um evento de mundo, conforme mostram as Figuras 4.29 e 4.30.

	EVENT_PROPERTY		
	ID	PROPERTY_OF	
Tolstoi's Participation in his birth	75	77	Tolstoi's Birth
Tolstoi's Birth Date	78	77	Tolstoi's Birth
Toltoi's Name Participation in its creation	82	81	Name Creation Event
Toistoi's Name Creation Event Date	83	81	Name Creation Event

Figura 4.29: Inclusão de propriedades de eventos em *DB:EventProperty*

	EVENT_ATTRIBUTION							
	ENTITY_ID	LOG_ID	TEXT_VALUE	DATE_VALUE	INT_VALUE	BOOLEAN_VALUE	REFERENCE	
Tolstoi's Participation in his birth	75	1					74	Liev Tolstoi
Tolstoi's Birth Date	78	1		09/09/1828				
Toltoi's Name Participation in its creation	84	1					79	Toltoi's Name
Toistoi's Name Creation Event Date	83	1		22/11/2019				

Figura 4.30: Atribuição de valores e referências a eventos

As Figuras 4.31 à 4.34 apresentam as tuplas resultantes da representação em banco de dados das entidades da Figura 4.24. A Figura 4.31 representa as tuplas relacionadas à inclusão de Sophia Behrs, seu evento de nascimento e seu nome. Por ser similar à representação de Liev Tolstoi, o estado das demais tabelas relacionadas foi omitido. A Figura 4.32 apresenta o *relator Casamento de Tolstoi e Sophia*, os *qua indivíduos* inerentes aos nubentes e partes do relator casamento, a participação do casamento no evento de casamento e o próprio evento de casamento. A Figura 4.33 apresenta os *qua indivíduos TolstoiQuaMarido* como inerente a Tolstoi e referenciando o relator *Casamento de Tolstoi e Sophia* e *SophiaQuaEsposa* como inerente a Sophia e também referenciando seu casamento. Por fim, a Figura 4.34 apresenta o evento de casamento e suas propriedades.

	ENTITY				
	ID	INSTANCE_OF	LOG_ID	HASH	INSTANCE_OF_DESC
Sophia Behrs	98	68	3		Woman
Sophia's Participation in her birth	99	66	3		HumanBirthParticipation
Sophia qua Preferred Named Object	100	58	3		PreferredNamedObject
Sophia's Birth	102	65	3		BirthEvent
Sophia's Birth Date	103	64	3		BirthDate
Sophia's preferred term	104	56	3		PreferredTerm
Sophia's preferred term name	105	62	3		TermName
Sophia's preferred term creation	110	28	3		CreationEvent
Sophia's preferred term in its creation	111	47	3		EndurantCreation
Sophia's preferred term creation date	112	45	3		EventDate

Figura 4.31: Tabela *DB:Entity* com detalhes sobre o indivíduo Sophia Behrs

	ENTITY				
	ID	INSTANCE_OF	LOG_ID	HASH	INSTANCE_OF_DESC
Sophia and Toltoi's Marriage	120	70	4		Marriage
Tolstoi qua Husband	121	69	4		Husband
Marriage participation in Sophia's wedding	122	72	4		MarriageStablishement
Sophia qua Wife	123	71	4		Wife
Sophia and Toltoi's Wedding	124	73	4		Wedding
Sophia and Toltoi's Wedding Date	125	45	4		EventDate

Figura 4.32: Casamento de Sophia e Liev Tolstoi

Além dos eventos de mundo, ontológicos, as tuplas inseridas nas tabelas *DB:Entity*, *DB:Attribution* e *DB:EventAttribution* possuem uma coluna referenciando eventos de sistemas: a coluna *log*. Eventos de sistema representam eventos de interação com o sistema AOM, ou seja, cadastro, alteração e correção. Interações que resultem em modificação do

	PROPERTY		
	ENTITY_ID	PROPERTY_OF	
Tolstoi qua Husband	121	74	Tolstoi
Sophia qua Wife	123	98	Sophia

	ATTRIBUTION					
	ENTITY_ID	EVENT	LOG_ID	...	REFERENCE	
Tolstoi qua Husband	121	124	4	...	120	Sophia and Toltoi's Marriage
Sophia qua Wife	123	124	4	...	120	Sophia and Toltoi's Marriage
		Sophia and Toltoi's Wedding				

Figura 4.33: Qua Individuals que participam do *relator* Casamento de Sophia e Tolstoi

	EVENT_PROPERTY	
	ENTITY_ID	PROPERTY_OF
Marriage participation in Sophia's wedding	122	124
Sophia and Toltoi's Wedding Date	125	124

	EVENT_ATTRIBUTION					
	ENTITY_ID	LOG_ID	DATE_VALUE	...	REFERENCE	
Marriage participation in Sophia's wedding	122	4		...	120	Sophia and Toltoi's Marriage
Sophia and Toltoi's Wedding Date	125	4	23/09/1862	...		

Figura 4.34: Propriedades do evento de casamento entre Sophia e Tolstoi

estado de banco de dados. Eles são particularmente importantes para diferenciar inclusão, alteração e correção.

A inclusão implica em inserir uma atribuição a uma propriedade que não possua valor. Essa atribuição é indexada por um evento de mundo e por um evento de sistema.

A alteração implica que existe um evento de mundo que altera, na realidade, a atribuição de valor de determinada propriedade. Por exemplo, em 1950, a capital brasileira era a cidade do Rio de Janeiro. Por determinação da lei 3.273 de 1957, a capital brasileira passa a ser Brasília desde 21 de abril de 1960, data de sua inauguração [119]. Logo, um evento de mundo, a inauguração de Brasília, mudou uma propriedade do país Brasil.

Um evento de sistema, epistemológico, por outro lado, ocorre quando há mudança de uma propriedade apenas em sistema, isto é, na realidade não houve qualquer evento que provocasse a mudança. Por exemplo, imagine que João da Silva, nascido em 23 de maio de 1955, fosse cadastrado em sistema com data de nascimento em 23 de maio de 1959. Ao se perceber que essa data foi cadastrada incorretamente, ela pode ser corrigida sem alteração do evento de mundo, ontológico, apenas alterando-se o evento de sistema, epistemológico. Ou seja, não houve nenhum evento na realidade que alterasse a data do nascimento de João da Silva, até porque eventos são imutáveis, entretanto houve um evento de sistema corrigindo seu valor.

A Figura 4.35 apresenta um exemplo de inclusão, alteração e correção em banco de dados.

ATTRIBUTION								
ENTITY_ID	EVENT_PARTICIPATION	LOG_ID	TEXT_VALUE	LONG_TEXT_VALUE	INT_VALUE	DATE_VALUE	BOOLEAN_VALUE	REFERENCE
129	130	5	Community property					
129	130	6	Separation of Property					
129	132	7	Partial community property					

Figura 4.35: Exemplo de inclusão, alteração e correção

A propriedade de id 129 representa o regime de bens de um determinado casamento. A atribuição do valor "Community Property" (Comunhão de bens) é indexada pelo evento de id 130 (coluna eventParticipation) no registro cinco (logId = 5). O registro seguinte (logId = 6) mostra uma modificação da atribuição em que o evento continua o mesmo, indicando que houve uma correção para o valor "Separation of Property". Ou seja, na realidade não houve mudança no regime de bens, mas ocorreu uma modificação de sistema, normalmente indicando que o primeiro cadastro foi feito equivocadamente. Desta forma, modificou-se apenas aspectos epistemológicos, isto é, modificou-se apenas o que se sabe sobre a realidade.

Por fim, a última atribuição do exemplo (logId = 7) apresenta uma alteração da atribuição na realidade, pois além de um novo registro (logId) houve também um novo evento da realidade indicando a alteração da atribuição: a atribuição não é mais indexada pelo evento de id 130 e, sim, pelo evento de id 132. Essa alteração na realidade ocorreria, por exemplo, caso os cônjuges alterassem o regime de divisão de bens do casamento (assumindo que isto seja possível). No primeiro caso, o regime de bens continuou o mesmo, mas o que se sabia sobre o regime de bens mudou.

Os eventos epistemológicos (registros) são representados na tabela *DB:Log*, conforme mostra a Figura 4.36.

LOG			
LOG_ID	TS	AGENT	DESCRIPTION
1	22/11/2019 21:46:50,341	jsmith	Inclusion
2	22/11/2019 22:03:59,275	jsmith	Change
3	22/11/2019 22:06:40,830	jsmith	Inclusion
4	22/11/2019 22:12:46,838	jsmith	Inclusion
5	22/11/2019 22:27:36,557	jsmith	Inclusion
6	22/11/2019 22:40:58,261	jsmith	Correction
7	22/11/2019 22:46:24,362	jsmith	Change

Figura 4.36: Conteúdo da tabela *DB:Log*

4.7 Restrições semânticas

As restrições apresentadas nesta Seção são implementadas utilizando o paradigma de banco de dados dedutivos [120]. Neste paradigma, as restrições podem ser implementadas por meio de projeções (*views*). As *views* contém uma consulta positiva que deve sempre estar vazia. Se uma *view* de integridade possui dados, significa que uma restrição não foi atendida [30, p. 274].

Para implementar as restrições tratadas na Seção 2.8 e na Seção 2.9, utilizou-se de uma tabela derivada de *DB:Generalization* chamada *Transitive_Generalization*.

Essa tabela é calculada de modo a exibir todas as especializações próprias possíveis entre tipos. Assim, por exemplo, Se A especializa B e B especializa C, então A especializa C. Na tabela *DB:Generalization* só duas tuplas são representadas: A especializando B e B especializando C. Em *Transitive_Generalization*, existem três tuplas: A especializando B, B especializando C e A especializando C.

4.7.1 Restrições UFO

As restrições UFO apresentadas na Seção 2.8 implementadas por meio de banco de dados dedutivos são apresentadas a seguir agrupadas pelas categorias UFO a que pertencem.

Substance Sortal e Kind

Restrição 4.7.1.1. Todo *objeto substancial* deve ser uma instância de um *substanceSortal*, direta ou indiretamente. Isso significa que todo elemento concreto do banco de dados deve estar numa hierarquia que inclui uma categoria de domínio que seja um *kind*, um *quantity* ou um *collective*;

A definição de *objeto substancial*, em banco de dados, implica na instanciação de uma das categorias citadas na Restrição 4.7.1.1. Logo, nenhuma implementação é realizada para esta restrição.

Restrição 4.7.1.2. Um *objeto substancial* não deve instanciar mais do que um *substanceSortal*;

O modelo físico de banco de dados não permite instanciar mais de uma categoria, logo a Restrição 4.7.1.2 não necessita implementação.

Restrição 4.7.1.3. Um *substanceSortal* não pode incluir outro *substanceSortal* nem um *subkind* (Subseção 2.8.3) em sua hierarquia de tipos, ou seja, um *substanceSortal* não pode ter um supertipo membro de { *kind*, *subkind*, *quantity*, *collective* };

```
WITH SORTAL AS (
    SELECT T.ID FROM ENTITY I
        INNER JOIN TYPE T
        ON I.ID = T.ID
        WHERE I.INSTANCE_OF IN (
            SELECT T2.ID FROM TYPE T2
            WHERE T2.DESCRIPTION = 'SortalType')
        OR
        I.INSTANCE_OF IN (
            SELECT S1.SUB_TYPE FROM TRANSITIVE_GENERALIZATION S1
            INNER JOIN TYPE T3
            ON S1.SUPER_TYPE = t3.ID
            WHERE t3.DESCRIPTION = 'SortalType')
), ULTIMATE_SORTAL AS (
    SELECT T.ID FROM ENTITY I
        INNER JOIN TYPE T
        ON I.ID = T.ID
        WHERE I.INSTANCE_OF IN (
            SELECT T2.ID FROM TYPE T2
            WHERE T2.DESCRIPTION = 'Kind'
            OR T2.DESCRIPTION = 'Collective'
            or T2.DESCRIPTION = 'Quantity')
), SUBKIND AS (
    SELECT T.ID FROM ENTITY I
        INNER JOIN TYPE T
        ON I.ID = T.ID
        WHERE I.INSTANCE_OF IN (
            SELECT T2.ID FROM TYPE T2
            WHERE T2.DESCRIPTION = 'Subkind')
        OR
        I.INSTANCE_OF IN (
            SELECT S1.SUB_TYPE FROM TRANSITIVE_GENERALIZATION S1
            INNER JOIN TYPE T3
            ON S1.SUPER_TYPE = t3.ID
            WHERE t3.DESCRIPTION = 'Subkind'))

SELECT ST.SUB_TYPE, T1.DESCRIPTION AS SUB_DESC, ST.SUPER_TYPE, T2.
DESCRIPTION AS SUPER_DESC
FROM TRANSITIVE_GENERALIZATION ST
INNER JOIN TYPE T1
ON T1.ID = ST.SUB_TYPE
INNER JOIN TYPE T2
ON T2.ID = ST.SUPER_TYPE
WHERE ST.SUPER_TYPE IN (SELECT ID FROM SORTAL)
```

```
44      AND ST.SUB_TYPE IN (SELECT ID FROM ULTIMATE_SORTAL)
```

Código 4.1: Código da *view* para verificação da Restrição 4.7.1.3

Restrição 4.7.1.4. Um tipo que represente um *universal substancial rígido* (*rigidSortal*) não pode ser uma subtipo de um tipo que representa um *universal anti-rígido*. Assim, um *substanceSortal* não pode ter como supertipo um membro de { *phase*, *role*, *roleMixin* }.

```
1       WITH ANTI_RIGID_TYPE AS (
2           SELECT T.ID FROM ENTITY I
3               INNER JOIN TYPE T
4               ON I.ID = T.ID
5               WHERE I.INSTANCE_OF IN (
6                   SELECT T2.ID FROM TYPE T2
7                   WHERE T2.DESCRIPTION IN ('RoleMixinType', 'RoleType', '
    PhaseType')
8               )
9               OR
10              I.INSTANCE_OF IN (
11                  SELECT S1.SUB_TYPE FROM TRANSITIVE_GENERALIZATION S1
12                  INNER JOIN TYPE t3
13                  ON S1.SUPER_TYPE = t3.ID
14                  WHERE t3.DESCRIPTION IN ('RoleMixinType', 'RoleType', '
    PhaseType')
15              )
16          ),
17          RIGID_TYPE AS (
18          SELECT T.ID FROM ENTITY I
19              INNER JOIN TYPE T
20              ON I.ID = T.ID
21              WHERE I.INSTANCE_OF IN (
22                  SELECT T2.ID FROM TYPE T2
23                  WHERE T2.DESCRIPTION IN ('Kind', 'Subkind')
24              )
25              OR
26              I.INSTANCE_OF IN (
27                  SELECT S1.SUB_TYPE FROM TRANSITIVE_GENERALIZATION S1
28                  INNER JOIN TYPE t3
29                  ON S1.SUPER_TYPE = t3.ID
30                  WHERE t3.DESCRIPTION IN ('Kind', 'Subkind')
31              )
32          )
33
34      SELECT TI.DESCRIPTION, RIGID.SUB_TYPE AS RIGID, RIGID.SUPER_TYPE AS
    ANTIRIGID
```

```
35      FROM TRANSITIVE_GENERALIZATION RIGID
36      INNER JOIN TYPE TI
37      ON RIGID.SUB_TYPE = TI.ID
38      WHERE RIGID.SUB_TYPE IN (SELECT ID FROM RIGID_TYPE)
39      AND RIGID.SUPER_TYPE IN (SELECT ID FROM ANTI_RIGID_TYPE)
```

Código 4.2: Código da view para verificação da Restrição 4.7.1.4

Subkinds

Restrição 4.7.1.5. Todo *subkind* deve estender um *kind*.

```
1   WITH KINDS AS (
2       SELECT I.ID FROM ENTITY I
3       WHERE I.INSTANCE_OF IN (
4           -- Seleciona o ID de kind
5           SELECT ID FROM TYPE
6           WHERE DESCRIPTION = 'Kind')
7   )
8
9   SELECT MULTI_KINDS.SUB_TYPE, T1.DESCRIPTION AS SUB_TYPE_DESC, ST.SUPER_TYPE
        , T2.DESCRIPTION AS SUPER_TYPE_DESC
10  FROM (SELECT SUB_TYPE, COUNT(SUB_TYPE) AS C
11          FROM TRANSITIVE_GENERALIZATION
12          WHERE SUB_TYPE IN (
13              SELECT I.ID FROM ENTITY I
14              WHERE I.INSTANCE_OF IN (
15                  SELECT ID FROM TYPE
16                  WHERE DESCRIPTION = 'Subkind'))
17          AND SUPER_TYPE IN (SELECT * FROM KINDS)
18          GROUP BY SUB_TYPE) MULTI_KINDS
19  INNER JOIN TRANSITIVE_GENERALIZATION ST
20  ON MULTI_KINDS.SUB_TYPE = ST.SUB_TYPE
21  INNER JOIN TYPE T1
22  ON MULTI_KINDS.SUB_TYPE = T1.ID
23  INNER JOIN TYPE T2
24  ON ST.SUPER_TYPE = T2.ID
25  WHERE MULTI_KINDS.C > 1
26  AND ST.SUPER_TYPE IN (SELECT * FROM KINDS)
```

Código 4.3: Código da view para verificação da Restrição 4.7.1.5

Abstract Mixin Type

Restrição 4.7.1.6. Tipos que representam *nonSortal* não podem ser subtipos de um tipo que representa um *sortal*. Como consequência, um *abstract mixin type* não pode ter como tipo superior um membro de { *kind*, *quantity*, *collective*, *subkind*, *phase*, *role* };

```
WITH ABSTRACT_MIXINS AS (
    SELECT I.ID, T.DESCRIPTION FROM ENTITY I
        INNER JOIN TYPE T
        ON I.ID = T.ID
        WHERE I.INSTANCE_OF IN (
            SELECT T2.ID FROM TYPE T2
            WHERE T2.DESCRIPTION = 'AbstractMixinType'
        )
        OR
        I.INSTANCE_OF IN (
            SELECT ST.SUB_TYPE FROM TRANSITIVE_GENERALIZATION ST
            INNER JOIN TYPE t3
            ON ST.SUPER_TYPE = t3.ID
            WHERE t3.DESCRIPTION = 'AbstractMixinType'
        )),
SORTALS AS (
    SELECT I.ID, T.DESCRIPTION FROM ENTITY I
        INNER JOIN TYPE T
        ON I.ID = T.ID
        WHERE I.INSTANCE_OF IN (
            SELECT T2.ID FROM TYPE T2
            WHERE T2.DESCRIPTION = 'SortalType'
        )
        OR
        I.INSTANCE_OF IN (
            SELECT S1.SUB_TYPE FROM TRANSITIVE_GENERALIZATION S1
            INNER JOIN TYPE t3
            ON S1.SUPER_TYPE = t3.ID
            WHERE t3.DESCRIPTION = 'SortalType'
        )
)


SELECT AM.ID AS ABSTRACT_MIXIN, AM.DESCRIPTION AS ABSTRACT_MIXIN_DESC, T1.
    ID AS SORTAL, T1.DESCRIPTION AS SORTAL_DESC
FROM ABSTRACT_MIXINS AM
INNER JOIN TRANSITIVE_GENERALIZATION ST
ON AM.ID = ST.SUB_TYPE
INNER JOIN TYPE T1
```

```
39 ON ST.SUPER_TYPE = T1.ID
40 WHERE ST.SUPER_TYPE IN (
41     SELECT ID FROM SORTALS)
```

Código 4.4: Código da view para verificação da Restrição 4.7.1.6

Restrição 4.7.1.7. Qualquer tipo que instanciar algum dos *nonSortal* não pode ter instâncias diretas. Dessa forma, qualquer instância de um dos *mixin class* não pode ter instâncias diretas.

```
1  WITH ABSTRACT_MIXINS AS (
2      SELECT I.ID, T.DESCRIPTION FROM ENTITY I
3          INNER JOIN TYPE T
4          ON I.ID = T.ID
5          WHERE I.INSTANCE_OF IN (
6              SELECT T2.ID FROM TYPE T2
7              WHERE T2.DESCRIPTION = 'AbstractMixinType'
8          )
9          OR
10         I.INSTANCE_OF IN (
11             SELECT ST.SUB_TYPE FROM TRANSITIVE_GENERALIZATION ST
12             INNER JOIN TYPE t3
13             ON ST.SUPER_TYPE = t3.ID
14             WHERE t3.DESCRIPTION = 'AbstractMixinType'
15         ))
16
17 SELECT ID, INSTANCE_OF FROM ENTITY
18 WHERE INSTANCE_OF IN (SELECT ID FROM ABSTRACT_MIXINS)
```

Código 4.5: Código da *view* para verificação da Restrição 4.7.1.7

Category

Restrição 4.7.1.8. Pelo fato de representar um universal rígido (que não pode especializar ou restringir um tipo anti-rígido[1]), e também pela combinação das regras do Subseção 2.8.4, um *category* não pode ter um tipo superior que pertença ao conjunto { *kind*, *quantity*, *collective*, *subkind*, *phase*, *role*, *roleMixin* };

A Restrição 4.7.1.8 é alcançada por meio da implementação da Restrição 4.7.1.6 e da *view* seguinte.

```
1  WITH CATEGORIES AS (
2      SELECT I.ID, T.DESCRIPTION FROM ENTITY I
```

[1] [32, p. 103]

```
        INNER JOIN TYPE T
        ON I.ID = T.ID
        WHERE I.INSTANCE_OF IN (
            SELECT T2.ID FROM TYPE T2
            WHERE T2.DESCRIPTION = 'CategoryType'
        )
        OR
        I.INSTANCE_OF IN (
            SELECT ST.SUB_TYPE FROM TRANSITIVE_GENERALIZATION ST
            INNER JOIN TYPE t3
            ON ST.SUPER_TYPE = t3.ID
            WHERE t3.DESCRIPTION = 'CategoryType'
        )
),
RoleMixins AS (
    SELECT I.ID, T.DESCRIPTION FROM ENTITY I
        INNER JOIN TYPE T
        ON I.ID = T.ID
        WHERE I.INSTANCE_OF IN (
            SELECT T2.ID FROM TYPE T2
            WHERE T2.DESCRIPTION = 'RoleMixinType'
        )
        OR
        I.INSTANCE_OF IN (
            SELECT ST.SUB_TYPE FROM TRANSITIVE_GENERALIZATION ST
            INNER JOIN TYPE t3
            ON ST.SUPER_TYPE = t3.ID
            WHERE t3.DESCRIPTION = 'RoleMixinType'))

SELECT SUB_TYPE,SUPER_TYPE FROM TRANSITIVE_GENERALIZATION
WHERE SUB_TYPE IN (SELECT ID FROM CATEGORIES)
AND SUPER_TYPE IN (SELECT ID FROM ROLEMIXINS)
```

Código 4.6: Código da *view* para atender parcialmente a Restrição 4.7.1.8

Restrição 4.7.1.9. Um *category* nunca possui instâncias diretas;

```
WITH CATEGORIES AS (
    SELECT I.ID, T.DESCRIPTION FROM ENTITY I
        INNER JOIN TYPE T
        ON I.ID = T.ID
        WHERE I.INSTANCE_OF IN (
            SELECT T2.ID FROM TYPE T2
            WHERE T2.DESCRIPTION = 'CategoryType'
        )
```

```
        OR
        I.INSTANCE_OF IN (
            SELECT ST.SUB_TYPE FROM TRANSITIVE_GENERALIZATION ST
            INNER JOIN TYPE t3
            ON ST.SUPER_TYPE = t3.ID
            WHERE t3.DESCRIPTION = 'CategoryType'))

SELECT ID, INSTANCE_OF FROM ENTITY I
WHERE I.INSTANCE_OF IN (SELECT ID FROM CATEGORIES)
```

Código 4.7: Código da *view* para verificação da Restrição 4.7.1.9

Restrição 4.7.1.10. Um *category* só pode especializar outro *category* ou um *mixin*.

```
WITH CATEGORIES AS (
    SELECT I.ID, T.DESCRIPTION FROM ENTITY I
        INNER JOIN TYPE T
        ON I.ID = T.ID
        WHERE I.INSTANCE_OF IN (
            SELECT T2.ID FROM TYPE T2
            WHERE T2.DESCRIPTION = 'CategoryType'
        )
        OR
        I.INSTANCE_OF IN (
            SELECT ST.SUB_TYPE FROM TRANSITIVE_GENERALIZATION ST
            INNER JOIN TYPE t3
            ON ST.SUPER_TYPE = t3.ID
            WHERE t3.DESCRIPTION = 'CategoryType'
        )
),
MIXINS AS (
    SELECT I.ID, T.DESCRIPTION FROM ENTITY I
        INNER JOIN TYPE T
        ON I.ID = T.ID
        WHERE
        I.INSTANCE_OF IN (
            SELECT T2.ID FROM TYPE T2
            WHERE T2.DESCRIPTION = 'MixinType'
        )
        OR
        I.INSTANCE_OF IN (
            SELECT ST.SUB_TYPE FROM TRANSITIVE_GENERALIZATION ST
            INNER JOIN TYPE t3
            ON ST.SUPER_TYPE = t3.ID
            WHERE t3.DESCRIPTION = 'MixinType'
```

```
        )
)

SELECT TST.SUB_TYPE, T1.DESCRIPTION AS DESCRIPTION_INF, TST.SUPER_TYPE, T2.
    DESCRIPTION AS DESCRIPTION_SUP FROM GENERALIZATION TST
INNER JOIN TYPE T1
ON TST.SUB_TYPE = T1.ID
INNER JOIN TYPE T2
ON TST.SUPER_TYPE = T2.ID
WHERE TST.SUB_TYPE IN (SELECT ID FROM CATEGORIES)
AND TST.SUPER_TYPE NOT IN (
    SELECT ID FROM MIXINS
    UNION SELECT ID FROM CATEGORIES)
```

Código 4.8: Código para verificação da Restrição 4.7.1.10

Mixin

Restrição 4.7.1.11. Um *mixin* não pode ter um *roleMixin* como um tipo superior;

```
WITH MIXINS AS (
    SELECT I.ID, T.DESCRIPTION FROM ENTITY I
        INNER JOIN TYPE T
        ON I.ID = T.ID
        WHERE
        I.INSTANCE_OF IN (
            SELECT T2.ID FROM TYPE T2
            WHERE T2.DESCRIPTION = 'MixinType'
        )
        OR
        I.INSTANCE_OF IN (
            SELECT ST.SUB_TYPE FROM TRANSITIVE_GENERALIZATION ST
            INNER JOIN TYPE t3
            ON ST.SUPER_TYPE = t3.ID
            WHERE t3.DESCRIPTION = 'MixinType'
        )
),
RoleMixins AS (
    SELECT I.ID, T.DESCRIPTION FROM ENTITY I
        INNER JOIN TYPE T
        ON I.ID = T.ID
        WHERE I.INSTANCE_OF IN (
            SELECT T2.ID FROM TYPE T2
            WHERE T2.DESCRIPTION = 'RoleMixinType'
        )
```

```
        OR
        I.INSTANCE_OF IN (
            SELECT ST.SUB_TYPE FROM TRANSITIVE_GENERALIZATION ST
            INNER JOIN TYPE t3
            ON ST.SUPER_TYPE = t3.ID
            WHERE t3.DESCRIPTION = 'RoleMixinType'
        )
)

SELECT SUB_TYPE,SUPER_TYPE FROM TRANSITIVE_GENERALIZATION
WHERE SUB_TYPE IN (SELECT ID FROM MIXINS)
AND SUPER_TYPE IN (SELECT ID FROM ROLEMIXINS)
```

Código 4.9: Código para verificação da Restrição 4.7.1.11

Restrição 4.7.1.12. Um *mixin* nunca possui instâncias diretas;

```
WITH MIXINS AS (
    SELECT I.ID, T.DESCRIPTION FROM ENTITY I
        INNER JOIN TYPE T
        ON I.ID = T.ID
        WHERE
        I.INSTANCE_OF IN (
            SELECT T2.ID FROM TYPE T2
            WHERE T2.DESCRIPTION = 'MixinType'
        )
        OR
        I.INSTANCE_OF IN (
            SELECT ST.SUB_TYPE FROM TRANSITIVE_GENERALIZATION ST
            INNER JOIN TYPE t3
            ON ST.SUPER_TYPE = t3.ID
            WHERE t3.DESCRIPTION = 'MixinType'
        )
)

SELECT I.ID, T.ID AS TYPE_ID, T.DESCRIPTION FROM ENTITY I
INNER JOIN TYPE T
ON I.INSTANCE_OF = T.ID
WHERE INSTANCE_OF IN (SELECT ID FROM MIXINS)
```

Código 4.10: Código para verificação da Restrição 4.7.1.12

Moment Type

Restrição 4.7.1.13. Todo *moment type* deve (direta ou indiretamente) estar conectado a uma associação com pelo menos uma relação do tipo *characterization*.

A Restrição 4.7.1.13 é implementada apenas para *Intrinsic Moments Types*.

```
WITH INTRINSIC_MOMENTS AS (
    SELECT I.ID, T.DESCRIPTION FROM ENTITY I
        INNER JOIN TYPE T
        ON I.ID = T.ID
        WHERE
        I.INSTANCE_OF IN (
            SELECT T2.ID FROM TYPE T2
            WHERE T2.DESCRIPTION = 'IntrinsicMomentType'
        )
        OR
        I.INSTANCE_OF IN (
            SELECT ST.SUB_TYPE FROM TRANSITIVE_GENERALIZATION ST
            INNER JOIN TYPE t3
            ON ST.SUPER_TYPE = t3.ID
            WHERE t3.DESCRIPTION = 'IntrinsicMomentType'
        )
)

SELECT ID,DESCRIPTION
FROM INTRINSIC_MOMENTS M
WHERE M.ID NOT IN (
    SELECT P.ID FROM PROPERTY_TYPE P
)
```

Código 4.11: Código para verificação dos *Intrinsic Moments* da Restrição 4.7.1.13

Mode Type

Restrição 4.7.1.14. Toda instância de um modeUniversal é existencialmente dependente (existentialDependence) de exatamente uma entidade.

```
SELECT ID, COUNT(ID) AS COUNT
FROM PROPERTY_TYPE
GROUP BY ID
HAVING COUNT(ID) < 1
```

Código 4.12: Código para verificação da Restrição 4.7.1.14

Relator

Restrição 4.7.1.15. Todo *relator* deve (direta ou indiretamente) estar conectado a uma associação com pelo menos uma relação *mediation*;

Restrição 4.7.1.16. Todo relator deve mediar ao menos duas entidades diferentes, dessa forma, seja R uma categoria instância de *relator*; $\{C_1, \ldots, C_n\}$ um conjunto de *universais* relacionados a R via uma relação *mediation* (*universal* mediados por R); $lower_{C_i}$ sendo $(1 \leq i \leq n)$ o valor mínimo da restrição de cardinalidade da associação conectada à C_i na relação *mediation*. Dessa forma, tem-se a restrição:

$$(\sum_{i=1}^{n} lower_{c_i}) \geq 2$$

```
WITH RELATOR AS (
    SELECT I.ID, T.DESCRIPTION FROM AOM_ENTITY I
        INNER JOIN AOM_TYPE t
        ON i.ID = t.ID
        WHERE
        I.INSTANCE_OF IN (
            SELECT t2.ID FROM AOM_TYPE t2
            WHERE t2.DESCRIPTION = 'RelatorType'
        )
        OR
        I.INSTANCE_OF IN (
            SELECT st.SUB_TYPE FROM AOM_TRANSITIVE_GENERALIZATION st
            INNER JOIN AOM_TYPE t3
            ON st.SUPER_TYPE = t3.ID
            WHERE t3.DESCRIPTION = 'RelatorType'
        )
), QUA_INDIVIDUAL AS (
    SELECT I.ID, T.DESCRIPTION FROM AOM_ENTITY I
        INNER JOIN AOM_TYPE t
        ON i.ID = t.ID
        WHERE
        I.INSTANCE_OF IN (
            SELECT t2.ID FROM AOM_TYPE t2
            WHERE t2.DESCRIPTION = 'QuaIndividualType'
        )
        OR
        I.INSTANCE_OF IN (
            SELECT st.SUB_TYPE FROM AOM_TRANSITIVE_GENERALIZATION st
            INNER JOIN AOM_TYPE t3
            ON st.SUPER_TYPE = t3.ID
            WHERE t3.DESCRIPTION = 'QuaIndividualType'
        )
)

SELECT A.CODOMAIN, COUNT(*)
```

```
FROM AOM_ATTRIBUTE A
INNER JOIN AOM_TYPE T
ON T.ID = A.CODOMAIN
WHERE A.ID IN (SELECT ID FROM QUA_INDIVIDUAL)
GROUP BY A.CODOMAIN
HAVING COUNT(*) < 2
```

Código 4.13: Implementação do código para a restrição Restrição 4.7.1.15

Restrição 4.7.1.17. Além da restrição anterior, adicionalmente, ao menos dois *universais* mediados por um *relator type* devem ser diferentes. Em outras palavras, *relator types* devem estar relacionados a *qua individuals types* diferentes.

```
WITH RELATOR AS (
    SELECT I.ID, T.DESCRIPTION FROM AOM_ENTITY I
        INNER JOIN AOM_TYPE t
        ON i.ID = t.ID
        WHERE
        I.INSTANCE_OF IN (
            SELECT t2.ID FROM AOM_TYPE t2
            WHERE t2.DESCRIPTION = 'RelatorType'
        )
        OR
        I.INSTANCE_OF IN (
            SELECT st.SUB_TYPE FROM AOM_TRANSITIVE_GENERALIZATION st
            INNER JOIN AOM_TYPE t3
            ON st.SUPER_TYPE = t3.ID
            WHERE t3.DESCRIPTION = 'RelatorType'
        )
), QUA_INDIVIDUAL AS (
    SELECT I.ID, T.DESCRIPTION FROM AOM_ENTITY I
        INNER JOIN AOM_TYPE t
        ON i.ID = t.ID
        WHERE
        I.INSTANCE_OF IN (
            SELECT t2.ID FROM AOM_TYPE t2
            WHERE t2.DESCRIPTION = 'QuaIndividualType'
        )
        OR
        I.INSTANCE_OF IN (
            SELECT st.SUB_TYPE FROM AOM_TRANSITIVE_GENERALIZATION st
            INNER JOIN AOM_TYPE t3
            ON st.SUPER_TYPE = t3.ID
            WHERE t3.DESCRIPTION = 'QuaIndividualType'
        )
```

```
)

SELECT A.CODOMAIN, T.DESCRIPTION, A.ID, COUNT(*)
FROM AOM_ATTRIBUTE A
INNER JOIN AOM_TYPE T
ON T.ID = A.CODOMAIN
WHERE A.ID IN (SELECT ID FROM QUA_INDIVIDUAL)
AND A.CODOMAIN IN (
    SELECT DISTINCT A.CODOMAIN
    FROM AOM_ATTRIBUTE A
    WHERE A.CODOMAIN IN (SELECT ID FROM RELATOR)
)
GROUP BY A.CODOMAIN, T.DESCRIPTION, A.ID
HAVING COUNT(*) > 1
```

Código 4.14: Implementação em SQL para a restrição Restrição 4.7.1.17

4.7.2 Outras Restrições Implementadas

Restrição 4.7.2.1. O contradomínio de um *qua individual type* deve, necessariamente, ser um *relator type*.

```
    WITH RELATOR AS (
        SELECT I.ID, T.DESCRIPTION FROM AOM_ENTITY I
            INNER JOIN AOM_TYPE t
            ON i.ID = t.ID
            WHERE
            I.INSTANCE_OF IN (
                SELECT t2.ID FROM AOM_TYPE t2
                WHERE t2.DESCRIPTION = 'RelatorType'
            )
            OR
            I.INSTANCE_OF IN (
                SELECT st.SUB_TYPE FROM AOM_TRANSITIVE_GENERALIZATION st
                INNER JOIN AOM_TYPE t3
                ON st.SUPER_TYPE = t3.ID
                WHERE t3.DESCRIPTION = 'RelatorType'
            )
    ), QUA_INDIVIDUAL AS (
        SELECT I.ID, T.DESCRIPTION FROM AOM_ENTITY I
            INNER JOIN AOM_TYPE t
            ON i.ID = t.ID
            WHERE
            I.INSTANCE_OF IN (
```

```
                SELECT t2.ID FROM AOM_TYPE t2
                WHERE t2.DESCRIPTION = 'QuaIndividualType'
            )
            OR
            I.INSTANCE_OF IN (
                SELECT st.SUB_TYPE FROM AOM_TRANSITIVE_GENERALIZATION st
                INNER JOIN AOM_TYPE t3
                ON st.SUPER_TYPE = t3.ID
                WHERE t3.DESCRIPTION = 'QuaIndividualType'
            )
    )

    SELECT T.ID, T.DESCRIPTION FROM AOM_ATTRIBUTE A
    INNER JOIN AOM_TYPE T
    ON T.ID = A.ID
    WHERE A.ID IN (SELECT ID FROM QUA_INDIVIDUAL)
    AND A.CODOMAIN  NOT IN (SELECT ID FROM RELATOR)
```

Código 4.15: Código para verificação da Restrição 4.7.2.1

Restrição 4.7.2.2. O contradomínio de *qualities* deve, necessariamente, ser um *datatype*.

```
WITH QUALITY AS (
    SELECT I.ID, T.DESCRIPTION FROM ENTITY I
        INNER JOIN TYPE T
        ON i.ID = T.ID
        WHERE
        I.INSTANCE_OF IN (
            SELECT T2.ID FROM TYPE T2
            WHERE T2.DESCRIPTION = 'QualityType'
        )
        OR
        I.INSTANCE_OF IN (
            SELECT ST.SUB_TYPE FROM TRANSITIVE_GENERALIZATION ST
            INNER JOIN TYPE t3
            ON ST.SUPER_TYPE = t3.ID
            WHERE t3.DESCRIPTION = 'QualityType'
        )
), DATA_TYPE AS (
    SELECT I.ID, T.DESCRIPTION FROM ENTITY I
        INNER JOIN TYPE T
        ON I.ID = T.ID
        WHERE
        I.INSTANCE_OF IN (
            SELECT T2.ID FROM TYPE T2
```

```
            WHERE T2.DESCRIPTION = 'Datatype'
        )
)

SELECT T.DESCRIPTION FROM ATTRIBUTE A
INNER JOIN TYPE T
ON T.ID = A.ID
WHERE A.ID IN (SELECT ID FROM QUALITY)
AND A.CODOMAIN  NOT IN (SELECT ID FROM DATA_TYPE)
```

Código 4.16: Código para verificação da Restrição 4.7.2.2

Restrição 4.7.2.3. Propriedades de eventos só podem estar relacionadas a eventos na tabela *DB:EventProperty*.

```

    SELECT EVENT.ID, EVENT.EVENT_ID, T.DESCRIPTION AS DESCRIPTION_EVENT
    FROM EVENT_PROPERTY EVENT
    INNER JOIN ENTITY I
    ON EVENT.EVENT_ID = I.ID
    INNER JOIN TYPE T
    ON I.INSTANCE_OF = T.ID
    WHERE I.INSTANCE_OF NOT IN (
        SELECT TST.SUB_TYPE
        FROM TRANSITIVE_GENERALIZATION TST
        INNER JOIN TYPE M1
        ON TST.SUPER_TYPE = M1.ID
        WHERE M1.DESCRIPTION = 'Perdurant')

```

Código 4.17: *DB:EventProperty* só pode ter propriedade de eventos

Restrição 4.7.2.4. Tabela *DB:Property* só pode conter propriedades de endurantes.

```
    --Busca por moments de endurantes em perdurantes
    SELECT MOMENT.ID, MOMENT.PROPERTY_OF AS BEARER, T.DESCRIPTION AS
    DESCRIPTION_EVENT
    FROM PROPERTY MOMENT
    INNER JOIN ENTITY I
    ON MOMENT.PROPERTY_OF = I.ID
    INNER JOIN TYPE T
    ON I.INSTANCE_OF = T.ID
    WHERE I.INSTANCE_OF IN (
        SELECT TST.SUB_TYPE
        FROM TRANSITIVE_GENERALIZATION TST
```

```
11          INNER JOIN TYPE M1
12          ON TST.SUPER_TYPE = M1.ID
13          WHERE M1.DESCRIPTION = 'Perdurant')
14
```

Código 4.18: *DB:Property* só pode conter propriedades de endurantes

4.8 Estimativa de Tempo de Modelagem e de Alteração do Modelo de Objetos

De modo a exemplificar o ganho de tempo quando se altera o modelo de objetos quando comparadas a abordagens tradicionais de construção de software, as seguintes alterações de modelo foram realizadas.

Para a execução deste exemplo, utilizamos um arquivo OWL para armazenar o modelo conceitual do domínio. Do arquivo OWL, apenas a hierarquia de especializações e instanciação é utilizada. O arquivo OWL não é necessário para o correto funcionamento da arquitetura, mas foi uma opção viável para este trabalho.

Partindo-se de um arquivo em OWL que já contenha categorias da UFO, tipos de dados e um sistema de nomes, mas sem categorias de domínio, medimos o tempo decorrido ao realizar algumas alterações. O tempo total de cada alteração inclui os seguintes passos:

- alteração do arquivo OWL;
- execução de um programa que implementa o modelo lógico $\mathcal{L}1$ e inclui as categorias presentes no arquivo OWL em banco de dados;
- atualizar uma aplicação web de modo a refletir as alterações em banco e permitir o cadastro das categorias recém-modeladas.

Para este exemplo, as seguintes alterações foram feitas:

1. Acrescentar a categoria *Person*, instância de *Kind*, como especialização de Complexo Funcional: 39 segundos;
2. Criar um evento de criação e atribuí-lo à categoria *Person*: três minutos e 15 segundos. Esse passo inclui os seguinte subpassos:
   (a) criar um evento instanciável chamado *BirthEvent*;
   (b) criar um *quality* data para o evento. O *quality* foi denominado *BirthDate*;
   (c) informar a cardinalidade, o portador do *quality* e qual o contradomínio do mesmo;

(d) Criar uma propriedade indicando a participação da pessoa no evento de nascimento. A propriedade foi denominada *HumanBirth*;

(e) preencher cardinalidade, portador (*BirthEvent*) e informar o contradomínio (*Person*);

(f) *Person* deve especializar um mixin que classifica entidades que precisam ter o seu evento de criação informado.

3. Criar dois subkinds: *Man*, *Woman*: um minuto e 23 segundos;

4. Criar um *relator Marriage* que media *man* e *woman*: seis minutos e 21 segundos. Esse passo inclui os seguintes subpassos:

   (a) Indicar *Marriage* como uma entidade com nome inferido, ou seja, *Marriage* especializa o *mixin inferredNameObject*, categoria do sistema de nomes;

   (b) Criar dois *qua indivíduos*: *ManQuaHusband* e *ManQuaWife* (em geral, optamos por cadastrar os qua indivíduos apenas com o nome dos papeis, ou seja, *Husband* e *Wife*;

   (c) Preencher detalhes como descrição, cardinalidade dos *qua indivíduos*.

5. Criar a categoria de eventos *Wedding* e atribuí-lo à *Marriage*: três minutos e 27 segundos. Esse passo é similar ao segundo passo, exceto por não precisar de um *quality*, pois herdou o *quality* data padrão de todos os eventos.

4.9 Considerações Finais

Este Capítulo apresenta como o modelo de objetos são representados em banco de dados. Apresenta detalhadamente a inclusão de tipos de dados, Universais e Particulares da UFO, Categorias de Domínio, Sistema de Nomes e Objetos de Domínio.

Por fim, apresenta um exemplo de alteração do modelo de objetos em que o tempo de alteração e disponibilização da alteração é mensurado.

Capítulo 5

Conclusões

A motivação deste trabalho partiu da complexidade do domínio de normas jurídicas. Entretanto, ao decorrer do desenvolvimento da pesquisa, os resultados demonstraram a viabilidade de aplicação em outros domínios, embora não fosse escopo realizar uma análise abrangente, nem os modelar no contexto desta pesquisa.

O trabalho relaciona duas abordagens complementares: a construção de um AOM associado a um modelo de objetos baseado em UFO para explorar e unir os benefícios de ambos os instrumentos.

A pesquisa desenvolve uma arquitetura estável para construção de software em domínios complexos que não necessita de alteração de código-fonte nem de estrutura de banco de dados.

O Objetivo Geral deste trabalho, apresentado na Subseção 1.3.1, é atingido na medida em que se constrói um modelo de aplicação que interpreta um modelo de objetos baseado em UFO que contenha entidades do domínio. Além disso, o trabalho também cumpre os Objetivos Específicos apresentados na Subseção 1.3.2. Os objetivos específicos são alcançados da seguinte forma:

1. o objetivo específico 1 é alcançado por meio da construção de um modelo de aplicação baseado no quadrado aristotélico (Seção 2.3) e no dodecágono ontológico (Seção 2.4) capaz de interpretar as entidades da UFO e as entidades de domínio. O modelo de aplicação é apresentado na Seção 3.2;

2. na Subseção 3.2.3 e Subseção 3.2.4, apresentam-se duas extensões ao modelo de aplicação para atender ao objetivo específico 3;

3. implementam-se restrições para atender aos axiomas das categorias da UFO utilizadas neste trabalho e, consequentemente, para cumprir com o objetivo específico 2.

No Capítulo 2, abordam-se conceitos importantes para a compreensão do trabalho, como metamodelagem, modelos adaptativos de objetos, ontologias, Ontologia, o quadrado aristotélico, o dodecágono ontológico, MLT e UFO.

No Capítulo 3, apresenta-se uma arquitetura AOM como solução ao problema de pesquisa. Por ser um AOM, a arquitetura possui um modelo de aplicação e um modelo de objetos, sendo este último baseado UFO.

O Capítulo 4 apresenta um exemplo de aplicação do modelo de objetos em banco de dados e, também, uma sequência de restrições semânticas. Apresenta também uma estimativa de tempo para que alterações no modelo de objetos fiquem funcionais.

Os principais resultados derivam da produção de um modelo de aplicação que consegue representar modelos de objetos baseados em UFO. O modelo de aplicação possui potencial para representar modelos de objetos de domínios variados, para permitir a integração de modelos e para realizar alterações de modelagem em tempo de execução, sem alteração de código-fonte e sem alteração estrutural em banco de dados.

Já o modelo de objetos conserva parte da riqueza, da expressividade e do direcionamento ontológico da UFO. A UFO permite construir modelos de objetos ricos e mitiga eventuais problemas de atualização de modelagem, comuns em sistemas AOMs.

As restrições semânticas baseadas em UFO validam em tempo real as categorias de domínio em conformidade com axiomas do fragmento UFO utilizado. Logo, a interoperabilidade entre modelos de objetos distintos é mais natural, ontologicamente bem-fundamentada e menos propensa a erros.

Os resultados desta pesquisa possuem potencial para otimizar a tarefa de engenharia de ontologia no que tange ao desenvolvimento de sistemas finais a partir de ontologias. Também reduz lacunas entre arquitetos e desenvolvedores, problema frequente de sistemas AOM, pois a UFO auxilia no entendimento de abstrações e dos elementos do domínio do problema.

Por outro lado, a complexidade adicional para a construção de um mecanismo extra para interpretação do *modelo de objetos* continua existente, uma vez que ainda é necessário construir uma camada de indireção que interpreta um modelo conceitual geral (UFO) para só depois abordar problemas específicos do domínio. Por isso, questões relacionadas ao custo extra de desenvolvimento inicial de sistemas AOM ainda se aplicam. De toda forma, entende-se que a adoção de uma ontologia bem-fundamentada tem a vantagem de facilitar e agilizar a compreensão da camada conceitual geral pelos desenvolvedores, em detrimento de outro modelo sem os fundamentos ontológicos adequados.

5.1 Possibilidades de Trabalhos Futuros

Para contribuir com trabalhos futuros, apresentam-se algumas propostas de avanço na pesquisa atual.

Criar uma interface para cadastro de categorias dos modelos de objetos

Este trabalho não possui uma interface para cadastro de categorias de domínio. Para superar esta limitação, utiliza-se de um arquivo OWL que, ao ser modificado, necessita de tradução para banco de dados.

Eliminar a dependência do arquivo OWL ofereceria mais autonomia ao modelo adaptativo de objetos. Isso pode ser obtido com o desenvolvimento de uma interface que permita cadastrar e alterar categorias de domínio e/ou categorias UFO. A interface deve verificar as restrições sintáticas e semânticas apresentadas neste trabalho.

A solução desenvolvida nesta pesquisa valida as categorias presentes no arquivo OWL por meio de um tradutor para banco de dados. Nenhuma validação é feita em OWL. Isso permite a criação de categorias que violem as restrições sintáticas e semânticas, pois a validação é feita apenas no momento de tentativa de inserção na base de dados. Uma interface dedicada permite aplicar restrições em tela, em tempo real, agilizando o processo de alteração e apresentando informações mais intuitivas sobre violações.

Traduzir modelos conceituais criados com OntoUML diretamente para banco de dados

Utiliza-se OntoUML para construção de modelos conceituais ontologicamente bem fundamentados em UFO. Algumas ferramentas permitem a criação de modelos em OntoUML, como apresentado em Benevides e Guizzardi [121] e em Guerson et al. [122].

Uma lacuna deixada por esta pesquisa é o mapeamento de modelos conceituais construídos em OntoUML para os modelos $\mathcal{L}1$ e de banco de dados. Trabalhos futuros podem explorar esta lacuna. Isso permitiria, por exemplo, obter um protótipo funcional apenas com a produção do modelo conceitual OntoUML, unir as restrições implementadas nas ferramentas OntoUML com as restrições apresentadas neste trabalho dentre outros.

Expor modelos de objetos presentes no banco de dados para modelos OntoUML

A possibilidade de exportar modelos persistidos em banco de dados para modelos conceituais em OntoUML automaticamente também não foi explorada nesta pesquisa e pode ser objeto de trabalhos futuros. Isso também permitiria a junção dos benefícios da abordagem desta pesquisa com os benefícios da OntoUML e suas ferramentas, obtendo assim, a partir do modelo de objetos, um modelo conceitual que pode ser manipulado em ferramentas de modelagem.

Aspectos Modais da UFO

Este trabalho não explorou as questões modais da semântica das entidades anti-rígidas da UFO-A, como *phases* e *roles*. Apesar do trabalho englobar *qua indivíduos*, não se explora a possibilidade de predicar sobre eles.

Por exemplo, o indivíduo *Maria* é instância do *Pessoa* e exerce dois papéis: o de *Estudante*, devido ao relacionamento material com a UnB mediado pelo *relator Matrícula de Maria na UnB*; o de *Esposa*, devido ao relacionamento material entre *Maria* e *João* mediado pelo *relator Casamento de Maria e João*. É possível representar que *Maria* participa nesses relacionamentos exercendo os papéis de *esposa* e *aluna*, porém não há distinção das ações e propriedades de *Maria* no exercício de tais papéis. Todas as ações e propriedades são atribuídas a Maria e não aos seus *qua indivíduos*.

Uma vantagem concreta desta abordagem pode ser mais bem compreendida com um exemplo real. *George Weah*, nascido em 1º de outubro de 1966 na Libéria, foi um jogador de futebol profissional que atuou por times como *Paris Saint-Germain* (PSG) e *Associazione Calcio Milan* (Milan). Porém, *Weah* também foi senador e hoje (em 2019) é presidente da Libéria [123].

Um portal de notícias, por exemplo, provavelmente classificaria uma notícia de acordo com o papel (*role*) exercido por *Weah*. Por exemplo, a notícia sobre a escolha de *Weah* como o melhor jogador do mundo em 1995 estaria em uma página de esportes. Já uma notícia sobre *Weah* como presidente é mais bem disponibilizada em uma página sobre política. *Weah* ainda é uma celebridade, logo notícias sobre sua vida pessoal possivelmente são melhor classificadas fora de páginas sobre política ou esporte.

O escopo desta pesquisa tratou da representação de *Weah* nos papeis citados, porém não abordou as ações ou propriedades de *Weah* no exercício de determinado papel. Em outras palavras, as propriedades de *WeahQuaJogador*, *WeahQuaPolítico*, *WeahQuaMarido* etc são atribuídos a *Weah* e não aos *qua indivíduos* inerentes a *Weah*.

Aperfeiçoamentos na lógica básica e na completude da ontologia de fundamentação

Além dos aspectos modais, pesquisas futuras podem explorar outras entidades da ontologia de fundamentação, considerando, por exemplo, avanços nas pesquisas que envolvem a formalização da UFO considerando lógicas adequadas para AOM.

No que tange à UFO-B, esta pesquisa limitou-se apenas a eventos *atômicos e instantâneos*. Desta forma, a representação de eventos como o *Torneio de Wimbledon de 2019*, a *Segunda Guerra Mundial*, a *Reforma Protestante* não são exploradas diretamente, apenas por meio de eventos *atômicos e instantâneos* que são parte de eventos complexos, como o *início do Torneio de Wimbledon de 2019*, o *fim da Segunda Guerra Mundial* ou a *publicação das 95 teses de Lutero em 31 de outubro de 1517*.

Por fim, trabalhos futuros podem explorar mais restrições da UFO de modo a atender a ontologia de fundamentação de forma mais abrangente.

Abordagem completa da MLT

Trabalhos futuros podem explorar uma abordagem mais sistemática e completa da MLT, visto que este trabalho atende parcialmente a teoria multinível e implementa apenas parte das suas restrições. Isso poderia trazer maior clareza conceitual e aprimorar a completude das verificações das restrições sintáticas mencionadas na Seção 3.3.

Avaliação empírica do modelo em produção

A pesquisa apresenta uma arquitetura para domínios complexos, entretanto apresentam-se exemplos simples de modo a proporcionar bom entendimento da arquitetura. Existe uma implementação similar a esta em produção, porém não há avaliação empírica sistemática dos resultados. Logo, avaliar empiricamente os resultados desta pesquisa pode ser objeto de trabalhos futuros.

Publicação dos resultados desta pesquisa em artigo científico

No momento de conclusão desta dissertação, os resultados ainda não haviam sido submetidos para publicação em artigo científico. Entretanto, uma submissão está planejada e há expectativa de que seja realizada no primeiro semestre de 2020.

Referências

[1] Colin Atkinson e T. Kuhne. Model-driven development: a metamodeling foundation. *IEEE Software*, 20(5):36–41, 9 2003. ISSN 0740-7459. doi: 10.1109/MS.2003.1231149. Disponível em: `http://ieeexplore.ieee.org/document/1231149/`. 2

[2] Grupo de Trabalho SILEX do Comitê Gestor de Informação do Portal LexML. Modelo de Requisitos para Sistemas Informatizados de Gestão da Informação Jurídica, 2013. Disponível em: `http://silex.lexml.gov.br`. 3

[3] Marco Antônio Filippi. Senado instala Centro de Informações. *Estado de São Paulo*, page 169, 3 1971. 3

[4] Gilberto Luiz do Amaral, João Eloi Olenike, Letícia M. Fernandes do Amaral, Cristiano Lisboa Yazbek, e Fernando Steinbruch. Quantidade de Normas Editadas no Brasil: 30 Anos da Constituição Federal de 1988. Technical report, IBPT – Instituto Brasileiro de Planejamento e Tributação, Curitiba, PR, 2018. 3, 4

[5] João Paulo Andrade Almeida. *Model-driven design of distributed applications*. PhD thesis, University of Twente, Netherlands, 3 2006. 4, 8, 16

[6] Oscar Pastor e Juan Carlos Molina. *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling*. Springer, 1 edition, 2007. ISBN 3540718672,9783540718673,9783540718680. 4, 8, 9, 16, 17

[7] Christian Plattner, Gustavo Alonso, e M Tamer Özsu. Extending DBMSs with satellite databases. *The VLDB Journal*, 17(4):657–682, 2008. 4

[8] Yannis Kalfoglou. *Cases on Semantic Interoperability for Information Systems Integration: Practices and Applications: Practices and Applications*. IGI Global, 2009. 4

[9] Philippe Kruchten. *The Rational Unified Process: An Introduction, Second Edition*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2000. ISBN 0201707101. 5

[10] Michael E Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182–211, 1976. 5

[11] Barry W Boehm e others. *Software engineering economics*, volume 197. Prentice-hall Englewood Cliffs (NJ), 1981. 5

[12] Research Triangle Institute. The economic impacts of inadequate infrastructure for software testing. Technical report, National Institute of Standards and Technology, 2002. Disponível em: `https://www.nist.gov/document/report02-3pdf`. 5

[13] Forrest Shull, Vic Basili, Barry Boehm, A Winsor Brown, Patricia Costa, Mikael Lindvall, Daniel Port, Ioana Rus, Roseanne Tesoriero, e Marvin Zelkowitz. What we have learned about fighting defects. In: *Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on*, pages 249–258, 2002. 5

[14] Barry Boehm e Richard Turner. *Balancing Agility and Discipline: A Guide for the Perplexed, Portable Documents*. Addison-Wesley Professional, 2003. 5

[15] Steve McConnell. *Code Complete, Second Edition*. Microsoft Press, Redmond, WA, USA, 2004. ISBN 0735619670, 9780735619678. 5

[16] Maurice Dawson, Darrell Norman Burrell, Emad Rahim, e Stephen Brewster. Integrating software assurance into the software development life cycle (SDLC). *Journal of Information Systems Technology & Planning*, 3(6):49–53, 2010. 5, 6

[17] PMI, editor. *Um Guia do Conhecimento em Gerenciamento de Projetos (Guia PMBOK)*. Project Management Institute, Newtown Square, Pennsylvania, 5 edition, 2013. ISBN 9781628250077. 5, 7

[18] Joseph W Yoder e Ralph Johnson. The Adaptive Object-Model Architectural Style. In: J. Bosch, M. Gentleman, C. Hofmeister, e J. Kuusela, editors, *Software Architecture*, volume 97 of *IFIP — The International Federation for Information Processing*, pages 3–27. Springer, Boston, MA, 2002. doi: 10.1007/978-0-387-35607-5{_}1. 6, 21

[19] Joseph W Yoder, Federico Balaguer, e Ralph Johnson. Architecture and design of adaptive object-models. *ACM SIGPLAN Notices*, 36(12):50–60, 2001. ISSN 03621340. doi: 10.1145/583960.583966. 6, 19, 20, 21

[20] Giancarlo Guizzardi. Ontology, Ontologies and the "I" of FAIR. *Data Intelligence*, 2(0):181–191, 2020. doi: 10.1162/dint{_}a{_}00040. Disponível em: `https://doi.org/10.1162/dint_a_00040`. 7

[21] Nicola Guarino e Pierdaniele Giaretta. Ontologies and Knowledge Bases Towards a Terminological Clarification. In: Nicolaas J.I. Mars, editor, *Towards very large knowledge bases: knowledge building & knowledge sharing*, pages 25–32. IOS Press, Amsterdam, The Netherlands, The Netherlands, 1995. ISBN 9051992173. 7

[22] Robert Audi, editor. *The Cambridge dictionary of philosophy*. ambridge University Press, Cambridge, UK, 2nd edition, 1999. ISBN 9780511074172. doi: 10.5860/choice.33-3059. 7

[23] Phillip Bricker. Ontological Commitment. In: Edward N Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 201 edition, 2016. 7

[24] Gyula Klima. The Medieval Problem of Universals. In: Edward N Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 201 edition, 2017. 8

[25] Michael J Loux e Thomas M Crisp. *Metaphysics: A contemporary introduction*. Routledge, 3rd edition, 2006. 8

[26] Gonzalo Rodriguez-Pereyra. Nominalism in Metaphysics. In: Edward N Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 201 edition, 2016. 9

[27] Giancarlo Guizzardi e Terry Halpin. Ontological Foundations for Conceptual Modelling. *Appl. Ontol.*, 3(1-2):1–12, 1 2008. ISSN 1570-5838. Disponível em: `http://dl.acm.org/citation.cfm?id=1412417.1412423`. 9

[28] Richard A Cohen. *Face to face with Levinas*. SUNY Press, 2012. 10

[29] Giancarlo Guizzardi, R A Falbo, e Renata Silva Souza Guizzardi. A importância de Ontologias de Fundamentação para a Engenharia de Ontologias de Domínio: o caso do domínio de Processos de Software. *Revista IEEE América Latina*, 6(3):244–251, 2008. 10

[30] Lauro César Araujo. *Uma linguagem para formalização de discursos com base em ontologias*. Coleção de teses, dissertações e monografias de servidores do Senado Federal. Senado Federal, Brasília, Brasil, 2017. ISBN 9788570189028. Disponível em: `http://www2.senado.leg.br/bdsf/item/id/543314`. 10, 12, 15, 27, 33, 36, 37, 46, 48, 106

[31] Michael Verdonck e Frederik Gailly. Insights on the Use and Application of Ontology and Conceptual Modeling Languages in Ontology-Driven Conceptual Modeling. In: Isabelle Comyn-Wattiau, Katsumi Tanaka, Il-Yeol Song, Shuichiro Yamamoto, e Motoshi Saeki, editors, *Conceptual Modeling: 35th International Conference, ER 2016, Gifu, Japan, November 14-17, 2016, Proceedings*, pages 83–97, Cham, 2016. Springer International Publishing. ISBN 978-3-319-46397-1. doi: 10.1007/978-3-319-46397-1{_}7. Disponível em: `https://doi.org/10.1007/978-3-319-46397-1_7`. 10, 11

[32] Giancarlo Guizzardi. *Ontological foundations for structural conceptual models*, volume 05-74. CTIT, Centre for Telematics and Information Technology, Enschede, The Netherlands, 2005. ISBN 9075176813. doi: 10.1007/978-3-642-31095-9{_}45. Disponível em: `http://doc.utwente.nl/50826`. 10, 27, 28, 31, 34, 35, 36, 37, 38, 39, 40, 46, 47, 48, 50, 53, 69, 70, 111

[33] Giancarlo Guizzardi. The role of foundational ontologies for conceptual modeling and domain ontology representation. In: *Databases and Information Systems, 2006 7th International Baltic Conference on*, pages 17–25, 2006. 10

[34] Renata Silva Souza Guizzardi. *Agent-oriented Constructivist Knowledge Management*. CTIT Ph.D.-thesis series. Centre for Telematics and Information Technology, University of Twente, 2006. ISBN 9789036523134. Disponível em: `https://books.google.com.br/books?id=JaOWkQEACAAJ`. 10

[35] Giancarlo Guizzardi e Gerd Wagner. Some applications of a unified foundational ontology in business modeling. In: *Business Systems Analysis with Ontologies*, pages 345–367. IGI Global, 2005. 10

[36] Yair Wand e Ron Weber. Mario Bunge's Ontology as a Formal Foundation for Information Systems Concepts. In: Paul Weingartner e Georg J W Dorn, editors, *Studies on Mario Bunge's Treatise*, Poznań studies in the philosophy of the sciences and the humanities. Rodopi, 1990. ISBN 9789051831870. Disponível em: `https://books.google.com.br/books?id=w-soo6UdwhAC`. 11

[37] Yair Wand e Ron Weber. On the deep structure of information systems. *Information Systems Journal*, 5, 1995. 11

[38] Ron Weber. *Ontological Foundations of Information Systems*. Coopers & Lybrand Research Methodology Monograph No. 4, Coopers & Lybrand, Melbourne, 1997. 11

[39] Wolfgang Degen, Barbara Heller, Heinrich Herre, e Barry Smith. GOL: A General Ontological Language. In: *FORMAL ONTOLOGY AND INFORMATION SYSTEMS*, 2001. 11, 28

[40] Barbara Heller e Heinrich Herre. Ontological categories in GOL. *Axiomathes*, 14 (1):57–76, 2004. 11, 28

[41] Owen Brady, Richard Overill, e Jeroen Keppens. Addressing the increasing volume and variety of digital evidence using an ontology. In: *Intelligence and Security Informatics Conference (JISIC), 2014 IEEE Joint*, pages 176–183, 2014. 11

[42] Owen Brady, Richard Overill, e Jeroen Keppens. DESO: Addressing volume and variety in large-scale criminal cases. *Digital Investigation*, 15:72–82, 2015. 11

[43] Nicola Guarino e Christopher Welty. Evaluating ontological decisions with OntoClean. *Communications of the ACM*, 45(2):61–65, 2002. 11

[44] Nicola Guarino e Christopher A Welty. An overview of OntoClean. In: *Handbook on ontologies*, pages 151–171. Springer, 2004. 11, 28

[45] Claudio Masolo, Stefano Borgo, Aldo Gangemi, Nicola Guarino, e Alessandro Oltramari. WonderWeb Deliverable D18 Ontology Library (final). Technical report, IST Project 2001-33052 WonderWeb: Ontology Infrastructure for the Semantic Web, 2003. 11

[46] Adam Pease, Ian Niles, e John Li. The suggested upper merged ontology: A large ontology for the semantic web and its applications. In: *Working notes of the AAAI-2002 workshop on ontologies and the semantic web*, volume 28, pages 7–10, 2002. 11

[47] Mario Bunge. *Ontology I: The Furniture of the World*, volume 3 of *Treatise on Basic Philosophy*. D. Reidel Publishing Company, Boston, MA, 1977. 11, 12

[48] Gove Allen e Salvatore March. A Critical Assessment of the Bunge-Wand-Weber Ontology for Conceptual Modeling. In: *SSRN Electronic Journal*, 3 2007. 12

[49] John Mylopoulos. Conceptual Modelling and Telos. In: Peri Loucopoulos e Roberto Zicari, editors, *Conceptual Modeling, Databases, and Case: An Integrated View of Information Systems Development*, chapter 2, pages 49–68. John Wiley & Sons, Inc., New York, NY, USA, 1992. 12

[50] Giancarlo Guizzardi, Gerd Wagner, Nicola Guarino, e Marten van Sinderen. An ontologically well-founded profile for UML conceptual models. In: *Advanced Information Systems Engineering*, pages 1–122, 2004. 12, 28, 30

[51] Giancarlo Guizzardi. Ontological meta-properties of derived object types. In: *International Conference on Advanced Information Systems Engineering*, pages 318–333, 2012. 12, 28, 30

[52] Giancarlo Guizzardi. Logical, ontological and cognitive aspects of object types and cross-world identity with applications to the theory of conceptual spaces. In: *Applications of Conceptual Spaces*, pages 165–186. Springer, 2015. 12, 28, 30

[53] Giancarlo Guizzardi. Modal Aspects of Object Types and Part-Whole Relations and the de re/de dicto Distinction. In: *Advanced Information Systems Engineering*, pages 5–20, 2007. 12, 28, 30, 41

[54] Giancarlo Guizzardi. The problem of transitivity of part-whole relations in conceptual modeling revisited. In: *International Conference on Advanced Information Systems Engineering*, pages 94–109, 2009. 12, 28, 30

[55] Giancarlo Guizzardi, Gerd Wagner, e Heinrich Herre. On the foundations of uml as an ontology representation language. In: *EKAW*, volume 3257, pages 47–62, 2004. 12, 28, 30

[56] Giancarlo Guizzardi e Veruska Zamborlini. Using a trope-based foundational ontology for bridging different areas of concern in ontology-driven conceptual modeling. *Science of Computer Programming*, 96:417–443, 2014. 12, 28, 30

[57] Giancarlo Guizzardi e Gerd Wagner. What's in a relationship: an ontological analysis. In: *International Conference on Conceptual Modeling*, pages 83–97, 2008. 12, 28, 30, 40

[58] Dolors Costal, Cristina Gómez, e Giancarlo Guizzardi. Formal semantics and ontological analysis for understanding subsetting, specialization and redefinition of associations in UML. In: *International Conference on Conceptual Modeling*, pages 189–203, 2011. 12, 28, 30

[59] Giancarlo Guizzardi. Agent Roles, Qua Individuals and The Counting Problem. In: P. Giorgini, A. Garcia, C. Lucena, e R. Choren, editors, *Software Engineering of Multi-Agent Systems*, pages 143–160. Springer-Verlag, 2006. 12, 28, 30

[60] Giancarlo Guizzardi, Gerd Wagner, João Paulo Andrade Almeida, e Renata Silva Souza Guizzardi. Towards Ontological Foundations for Conceptual Modeling: The Unified Foundational Ontology (UFO) Story. *Applied ontology*, 10(3-4): 259–271, 3 2015. 12, 28

[61] Fernando Carolo e Leonardo Burlamaqui. Improving WEB Content Management with Semantic Technologies. In: *SemTech 2011 – Semantic Technology Conference 2011*, San Francisco, 2011. Disponível em: `http://aflux.in/files/websemantica_globo.pdf`. 12

[62] Andréa C F Albuquerque, José L Campos dos Santos, e Alberto N de Castro. OntoBio: A Biodiversity Domain Ontology for Amazonian Biological Collected Objects. In: *System Sciences (HICSS), 2015 48th Hawaii International Conference on*, pages 3770–3779, 2015. 12

[63] Ricardo Werlang. *Ontology-based approach for standard formats integration in reservoir modeling*. PhD thesis, Universidade Federal do Rio Grande do Sul, 2015. Disponível em: `http://hdl.handle.net/10183/115196`. 12

[64] Bernardo Gonçalves, Giancarlo Guizzardi, e José G Pereira Filho. Using an ECG reference ontology for semantic interoperability of ECG data. *Journal of Biomedical Informatics*, 44(1):126–136, 2011. 12

[65] Matthias Steup. Epistemology. In: Edward N Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 201 edition, 2018. 13

[66] Eduardo Moresi. *Metodologia da pesquisa*. Universidade Católica de Brasília, Brasília, Brasil, 2003. 14

[67] Veruska Carretta Zamborlini. *Estudo de Alternativas de Mapeamento de Ontologias da Linguagem OntoUML Para OWL: Abordagens Para Representação de Informação Temporal*. PhD thesis, Universidade Federal do Espírito Santo, 2011. 15, 27, 29, 31, 32, 33, 34, 35, 36, 38, 39, 40, 59

[68] Giancarlo Guizzardi, João Paulo Andrade Almeida, Renata Silva Sousa Guizzardi, Monalessa Perini Barcellos, e Ricardo Falbo. Ontologias de Fundamentação, Modelagem Conceitual e Interoperabilidade Semântica. In: *Proceedings of the Iberoamerican Meeting of Ontological Research*, 2011. 15, 22, 30

[69] Paulo Sérgio Santos Jr, João Paulo Andrade Almeida, e Giancarlo Guizzardi. Uma interpretação para os elementos de EPCs com base em uma ontologia de fundamentação. In: *3rd Workshop on Business Process Modeling Management (WBPM)*, Fortaleza, Brasil, 2009. Disponível em: `https://inf.ufes.br/~gguizzardi/WBPM-Santos-Almeida-Guizzardi.pdf`. 15

[70] Aline Martins, Ricardo A Falbo, Giancarlo Guizzardi, e João Paulo Andrade Almeida. Uso de uma Ontologia de Fundamentação para Dirimir Ambiguidades na Modelagem de Processos de Negócio. In: *VII Simpósio Brasileiro de Sistemas de Informação (SBSI)*, 2011. 15

[71] Alex Pinheiro das Graças e Giancarlo Guizzardi. Padrões de Modelagem e Regras de Construção de Modelos para a criação de Ontologias de Domínio Bem-Fundamentadas em OntoUML. In: *3ª Ontobras - Seminário de Pesquisa em Ontologia no Brasil, Florianópolis*, page 10, Florianópolis, Brasil, 2010. Disponível em: `http://www.lbd.dcc.ufmg.br/colecoes/ontobras/2010/0030.pdf`. 15

[72] Stephen J Mellor, Anthony N Clark, e Takao Futagami. Guest Editors' Introduction: Model-Driven Development. *IEEE Softw.*, 20(5):14–18, 9 2003. ISSN 0740-7459. doi: 10.1109/MS.2003.1231145. Disponível em: `https://doi.org/10.1109/MS.2003.1231145`. 16

[73] H Stachowiak. *Allgemeine Modelltheorie*. Springer-Verlag, 1973. ISBN 9783211811061. Disponível em: `https://books.google.com.br/books?id=DK-EAAAAIAAJ`. 16

[74] Uwe Aßmann, Steffen Zschaler, e Gerd Wagner. Ontologies, meta-models, and the model-driven paradigm. In: *Ontologies for software engineering and software technology*, pages 249–273. Springer, 2006. 17, 18, 22

[75] Michael Pidd. *Tools for thinking: modelling in management science*. Wiley, 2 edition, 2003. ISBN 0470847956. 18

[76] Thomas Kühne. What is a Model? In: *Dagstuhl Seminar Proceedings*, 2005. 18

[77] Edwin Seidewitz. What models mean. *IEEE software*, 20(5):26–32, 2003. 18

[78] Anita K Jones. The object model: A conceptual tool for structuring software. In: R Bayer, R M Graham, e G Seegmüller, editors, *Operating Systems*, Lecture Notes in Computer Science, 60, chapter 2, pages 7–16. Springer, Berlin, Heidelberg, 1978. 19

[79] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William E Lorensen, e others. *Object-oriented modeling and design*. Prentice-hall Englewood Cliffs, NJ, 1st edition, 1991. 19

[80] R G G Cattell e D K Barry. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann series in data management systems. Morgan Kaufmann Publishers, 1997. ISBN 9781558604636. 19

[81] Dirk Riehle, Michel Tilman, e Ralph Johnson. Dynamic Object Model. In: Dragos-Anton Manolescu, Markus Voelter, e James Noble, editors, *Pattern Languages of Program Design*, volume 5 of *Software Patterns Series*, chapter 1, pages 3–24. Addison-Wesley Professional, 2000. doi: 10.1.1.203.697. Disponível em: `https://hillside.net/plop/plop2k/proceedings/Riehle/Riehle.pdf`. 19, 20

[82] Martin Fowler. *Analysis Patterns: Reusable Object Models*. Addison-Wesley Professional, 1996. ISBN 9780201895421,0201895420. 19

[83] Thomas Hofweber. Logic and Ontology. In: Edward N Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, fall 2014 edition, 2014. 21

[84] Maurício Barcellos Almeida. Uma abordagem integrada sobre ontologias: Ciência da Informação, Ciência da Computação e Filosofia. *Perspectivas em Ciência da Informação*, 19(3):242–258, 2014. Disponível em: `https://dx.doi.org/10.1590/1981-5344/1736`. 22, 23

[85] Edward Jonathan Lowe. *The Four-Category Ontology: A Metaphysical Foundation for Natural Science*. Clarendon Press, 2007. 22, 23, 24, 25, 28, 31

[86] Thomas R Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993. 22

[87] Roderick M Chisholm. The Basic Ontological Categories. In: Kevin Mulligan, editor, *Language, Truth and Ontology*, pages 1–13. Springer Netherlands, Dordrecht, 1992. ISBN 978-94-011-2602-1. doi: 10.1007/978-94-011-2602-1{_}1. Disponível em: `https://doi.org/10.1007/978-94-011-2602-1_1`. 23

[88] Merriam-Webster. Instantiate, 2018. Disponível em: `https://www.merriam-webster.com/dictionary/instantiate`. 25

[89] Giancarlo Guizzardi e Gerd Wagner. Towards a Logic of the Ontological Dodecagon. *From Philosophy to Computational Logic: Festschrift in Honour of David Pearce's First*, 60, 2012. 25, 26, 27, 65, 66, 67, 71

[90] Giancarlo Guizzardi. On ontology, ontologies, conceptualizations, modeling languages, and (meta) models. *Frontiers in artificial intelligence and applications*, 155:18, 2007. 27

[91] Giancarlo Guizzardi, Gerd Wagner, Ricardo De Almeida Falbo, Renata S.S. Guizzardi, e João Paulo Andrade Almeida. Towards ontological foundations for the conceptual modeling of events. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8217 LNCS, pages 327–341, 2013. ISBN 9783642419232. doi: 10.1007/978-3-642-41924-9{_}27. 28, 41, 42, 43, 44, 45, 46

[92] Giancarlo Guizzardi, Ricardo de Almeida Falbo, e Renata Silva Souza Guizzardi. Grounding Software Domain Ontologies in the Unified Foundational Ontology (UFO): The case of the ODE Software Process Ontology. In: *CIbSE*, pages 127–140, 2008. 28

[93] Renata Silva Souza Guizzardi e Giancarlo Guizzardi. Ontology-Based Transformation Framework from Tropos to AORML. In: Eric Yu, Paolo Giorgini, Neil Maiden, John Mylopoulos, e Stephen Fickas, editors, *Social Modeling for Requirements Engineering*, Cooperative information systems, pages 547–571. MIT Press, Cambridge, Massachusetts / London, England, 2011. ISBN 9780262240550. 28

[94] Wolfgang Degen, Barbara Heller, Heinrich Herre, e Barry Smith. GOL: Toward an Axiomatized Upper-level Ontology. In: *Proceedings of the International Conference on Formal Ontology in Information Systems - Volume 2001*, FOIS '01, pages 34–46, New York, NY, USA, 2001. ACM. ISBN 1-58113-377-4. doi: 10.1145/505168.505173. Disponível em: `http://doi.acm.org/10.1145/505168.505173`. 28

[95] Giancarlo Guizzardi e Gerd Wagner. Using the unified foundational ontology (UFO) as a foundation for general conceptual modeling languages. In: *Theory and Applications of Ontology: Computer Applications*, pages 175–196. Springer, 2010. 28, 30, 33, 36

[96] Joseph LaPorte. Rigid Designators. In: Edward N Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 201 edition, 2018. 29

[97] Amie L. Thomasson. *Fiction and Metaphysics*. Cambridge Studies in Philosophy. Cambridge University Press, Cambridge, UK, 1999. ISBN 9780521640800. doi: 10.1017/CBO9780511527463. Disponível em: `https://www.cambridge.org/core/product/identifier/9780511527463/type/book`. 30

[98] Oxford English Dictionary Online. Instance, 2018. Disponível em: `https://en.oxforddictionaries.com/definition/instance`. 32

[99] Giancarlo Guizzardi. On the Representation of Quantities and Their Parts in Conceptual Modeling. In: *Proceedings of the 2010 Conference on Formal Ontology in Information Systems: Proceedings of the Sixth International Conference (FOIS 2010)*, pages 103–116, Amsterdam, The Netherlands, The Netherlands, 2010. IOS Press. ISBN 978-1-60750-534-1. Disponível em: `http://dl.acm.org/citation.cfm?id=1804715.1804728`. 35, 41

[100] Giancarlo Guizzardi. Representing collectives and their members in UML conceptual models: an ontological analysis. In: *International Conference on Conceptual Modeling*, pages 265–274, 2010. 35

[101] Antônio Houaiss, Mauro de Salles Villar, e Francisco Manoel de Mello Franco. *Dicionário Houaiss da língua portuguesa*. Objetiva, 2009. ISBN 9788573029635. 40, 71

[102] Giancarlo Guizzardi. Ontological foundations for conceptual part-whole relations: the case of collectives and their parts. In: *International Conference on Advanced Information Systems Engineering*, pages 138–153, 2011. 41

[103] Alessander Botti Benevides, Jean-Rémi Bourguet, Giancarlo Guizzardi, Rafael Peñaloza, e João Paulo A. Almeida. Representing a Reference Foundational Ontology of Events in SROIQ. *Applied ontology*, 14(3):293–334, 2019. doi: https://dx.doi.org/10.3233/ao-190214. 41

[104] João Paulo A Almeida, Ricardo A Falbo, e Giancarlo Guizzardi. Events as Entities in Ontology-Driven Conceptual Modeling. In: *Forthcoming in Conceptual Modeling. ER 2019. Lecture Notes in Computer Science*, pages 469–483, 2019. doi: 10.1007/978-3-030-33223-5{_}39. Disponível em: `https://doi.org/10.1007/978-3-030-33223-5_39http://link.springer.com/10.1007/978-3-030-33223-5_39`. 42, 75, 76

[105] Ingvar Johansson. Qualities, Quantities, and the Endurant-Perdurant Distinction in Top-Level Ontologies. In: *3rd Conference Professional Knowledge Management WM 2005*, pages 543–550. Springer Verlag, 2005. 41

[106] Stephen Mumford. *Dispositions*. Oxford University Press, 1 2003. ISBN 9780199259823. doi: 10.1093/acprof:oso/9780199259823.001.0001. Disponível em: `http://www.oxfordscholarship.com/view/10.1093/acprof:oso/9780199259823.001.0001/acprof-9780199259823`. 45, 46

[107] João Paulo A. Almeida, Victorio A. Carvalho, Freddy Brasileiro, Claudenir M. Fonseca, e Giancarlo Guizzardi. Multi-level conceptual modeling: Theory and applications. *CEUR Workshop Proceedings*, 2228:26–41, 2018. ISSN 16130073. 53, 54, 55

[108] Victorio A. Carvalho e João Paulo A. Almeida. Toward a well-founded theory for multi-level conceptual modeling. *Software and Systems Modeling*, 17(1):205–231, 2018. ISSN 16191374. doi: 10.1007/s10270-016-0538-9. 54

[109] Victorio A. Carvalho, João Paulo A. Almeida, Claudenir M. Fonseca, e Giancarlo Guizzardi. Extending the Foundations of Ontology-Based Conceptual Modeling with a Multi-level Theory. In: Paul Johannesson, Mong Li Lee, Stephen W. Liddle, Andreas L. Opdahl, e Óscar Pastor López, editors, *Conceptual Modeling*, volume 9381, pages 119–133. Springer International Publishing, 2015. ISBN 9783319252636. doi: 10.1007/978-3-319-25264-3{_}9. Disponível em: `http://link.springer.com/10.1007/978-3-319-25264-3_9`. 55, 56

[110] Zdenek Rybola. *Towards OntoUML for Software Engineering: Transformation of OntoUML into Relational Databases*. PhD thesis, Czech Technical University in Prague, 2017. Disponível em: `https://www.fit.cvut.cz/sites/default/files/PhDThesis-Rybola.pdf`. 56

[111] Chengwei Zhang, Ling Tian, e Yuanhao Wu. Ontology-Based Adaptive Object Model for Simulation Physical Parameter Management. In: *Volume 11: Systems, Design, and Complexity*, volume 11 of *ASME International Mechanical Engineering Congress and Exposition*. American Society of Mechanical Engineers, 11 2015. ISBN 978-0-7918-5754-0. doi: 10.1115/IMECE2015-50188. Disponível em: `https://doi.org/10.1115/IMECE2015-50188https://asmedigitalcollection.asme.org/IMECE/proceedings/IMECE2015/57540/Houston,Texas,USA/256331`. 57

[112] Victorio A. Carvalho, João Paulo A. Almeida, e Giancarlo Guizzardi. Using reference domain ontologies to define the real-world semantics of domain-specific languages. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8484 LNCS, pages 488–502, 2014. ISBN 9783319078809. doi: 10.1007/978-3-319-07881-6{_}33. 61

[113] Itzel Morales-Ramirez, Anna Perini, e Renata Guizzardi. An ontology of online user feedback in software engineering. *Applied Ontology*, 10:297–330, 2015. doi: 10.3233/AO-150150. 62

[114] I Puja, P Poscic, e D Jaksic. Overview and Comparison of Several elational Database Modelling Metodologies and Notations. In: *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1641–1646. IEEE, 2019. 63

[115] Nicola Guarino, Giancarlo Guizzardi, e Tiago Prince Sales. On the ontological nature of REA core relations. *CEUR Workshop Proceedings*, 2239:89–98, 2018. ISSN 16130073. 69, 70

[116] Nicola Guarino e Giancarlo Guizzardi. We need to discuss the relationship: Revisiting relationships as modeling constructs. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2015. ISBN 9783319190686. doi: 10.1007/978-3-319-19069-3{_}18. 70

[117] Richard Cyganiak, David Wood, e Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax, 2014. Disponível em: `http://www.w3.org/TR/rdf11-concepts/`. 71

[118] Gove N. Allen e Salvatore T. March. The effects of state-based and event-based data representation on user performance in query formulation tasks. *MIS Quarterly: Management Information Systems*, 30(2):269–290, 2006. ISSN 02767783. doi: 10.2307/25148731. 75

[119] Estados Unidos do Brasil. Lei nº 3.273, de 1º de Outubro de 1957, 1957. Disponível em: `https://www.lexml.gov.br/urn/urn:lex:br:federal:lei:1957-10-01;3273`. 104

[120] Robert M Colomb. *Deductive Databases and Their Applications*. Taylor & Francis, Inc., Bristol, PA, USA, 1st edition, 1998. ISBN 0748407960. 106

[121] Alessander Botti Benevides e Giancarlo Guizzardi. A model-based tool for conceptual modeling and domain ontology engineering in OntoUML. In: *International Conference on Enterprise Information Systems*, pages 528–538. Springer, 2009. 126

[122] J Guerson, T P Sales, G Guizzardi, e J P A Almeida. OntoUML Lightweight Editor: A Model-Based Environment to Build, Evaluate and Implement Reference Ontologies. In: *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop*, pages 144–147, 2015. doi: 10.1109/EDOCW.2015.17. 126

[123] The Editors of Encyclopaedia Britannica. George Weah. In: *Encyclopædia Britannica*. Encyclopædia Britannica, inc., 2019. Disponível em: `https://www.britannica.com/biography/George-Weah`. 127