Félix Luiz Zanetti

Representação Ontológica de Frameworks de Mapeamento Objeto/Relacional

Vitória, ES

2020

Félix Luiz Zanetti

Representação Ontológica de Frameworks de Mapeamento Objeto/Relacional

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Programa de Pós-Graduação em Informática

Orientador: Prof. Dr. Vítor E. Silva Souza

Vitória, ES

2020

Félix Luiz Zanetti
Representação Ontológica de Frameworks de Mapeamento Objeto/Relacional/ Félix Luiz Zanetti. – Vitória, ES, 2020-
58 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Vítor E. Silva Souza

Dissertação de Mestrado – Universidade Federal do Espírito Santo – UFES
Centro Tecnológico
Programa de Pós-Graduação em Informática, 2020.

1. Ontologias. 2. Frameworks. 3. Mapeamento Objeto/Relacional. 4. Impedância Objeto/Relacional. I. Souza, Vítor Estêvão Silva. II. Universidade Federal do Espírito Santo. IV. Representação Ontológica de Frameworks de Mapeamento Objeto/Relacional

CDU 02:141:005.7

Félix Luiz Zanetti

Representação Ontológica de Frameworks de Mapeamento Objeto/Relacional

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Trabalho aprovado. Vitória, ES, 25 de setembro de 2014:

Prof. Dr. Vítor E. Silva Souza
Orientador

Profª. Drª. Rita C. Galarraga Berardi
Universidade Tecnológica Federal do Paraná

Profª. Drª. Monalessa Perini Barcellos
Universidade Federal do Espírito Santo

Vitória, ES
2020

Agradecimentos

A Deus, por permitir a vivência no dia-a-dia e a chegada até aqui.

Aos meus pais, Céia e Carlinhos, por serem meus primeiros professores e apoiadores incondicionais de toda e qualquer ideia que envolva a evolução da minha vida acadêmica e profissional.

Aos professores do Nemo por passarem o conhecimento necessário, em especial ao meu orientador Vítor, por tanta paciência e disposição e ao Falbo, por auxiliar na evolução do trabalho. Também aos colegas do Nemo pelas importantes contribuições, seja na pesquisa, papos para descontração ou no revezamento ao fazer café.

Não posso deixar de agradecer àqueles que tornaram confortável a vida em Vitória nesses dois anos: Ramon e Géssica, os membros do JOVENS, os companheiros de apartamento Mateus e Rodrigo, a galera do vôlei, os amigos das jogatinas e, claro, os amigos que sentaram em um bar para tomar uma cerveja e falar da vida.

É necessário incluir um registro de agradecimento também àqueles que há anos são meus amigos e confidentes de sonhos e angústias: Luis Felipe, Jordanna, Matheus, Camila, Marília, Mariane, André, Willyam e Higor.

Por fim, agradeço a todos os colegas do Ifes campus Montanha que, de certa forma, tornaram possível o afastamento que foi essencial para a realização desse trabalho.

Resumo

O surgimento do paradigma Orientado a Objetos e sua ampla adoção conjunta com o paradigma Relacional no desenvolvimento de software deu destaque ao problema da Impedância Objeto/Relacional, ocasionado pela diferença de abordagem entre os paradigmas. Hoje, a utilização de *frameworks* de mapeamento entre as duas abordagens é estado da prática devido à eficiência e segurança que proporcionam ao desenvolvimento.

Sendo assim, diferentes *frameworks* desse tipo para diferentes linguagens de programação foram sendo desenvolvidos. E, mesmo com uma sintaxe diferente em cada um deles, é possível notar uma semântica comum.

Diante disso, este trabalho apresenta a Ontologia de *Frameworks* de Mapeamento Objeto/-Relacional (*Object/Relational Mapping Ontology* – ORM-O), uma ontologia de referência no domínio de *frameworks* ORM que visa identificar e representar a semântica do mapeamento objeto/relacional.

A ontologia foi desenvolvida seguindo o método SABiO e modelada usando OntoUML. Sua avaliação foi realizada por meio de atividades de verificação e validação, respondendo às questões de competência levantadas previamente e instanciando conceitos da ontologia utilizando trechos de código de mapeamento objeto/relacional, usando um *framework* ORM bastante popular.

ORM-O foi construída no escopo de um projeto que visa criar uma rede de ontologias sobre *frameworks* de desenvolvimento de software. Tais ontologias nos permitirão automatizar tarefas de interoperabilidade semântica, como migração de código entre *frameworks*, ou ainda especificar *smells* na arquitetura do software de forma independente de *framework* ou linguagem, por exemplo.

Como prova de conceito, este trabalho apresenta uma ferramenta de migração desenvolvida que converte código de um *framework* ORM para outro (em plataformas diferentes), utilizando a ontologia como *interlingua*.

Palavras-chaves: Ontologias; Frameworks; Mapeamento Objeto/Relacional; Impedância Objeto/Relacional.

Abstract

The emergence of the Object Oriented and Relational paradigms and their widespread adoption in software development highlighted the Object/Relational Impedance Mismatch, caused by the difference in approach between the paradigms. Nowadays, the use of mapping frameworks between the two approaches is state-of-the-practice due to the efficiency and security they provide for development.

Thus, different frameworks of this type for different programming languages have been developed. And, even with a different syntax in each of them, it is possible to notice shared semantics.

Therefore, in this work, we present the Object/Relational Mapping Ontology (ORM-O), a reference ontology in the ORM frameworks domain that aims to identify and represent the semantics of object/relational mapping.

The ontology was developed following the SABiO method and modeled using OntoUML. Its evaluation was carried out through verification and validation activities, answering the competency questions previously raised and instantiating ontology concepts using object/relational mapping code snippets, using a very popular ORM framework.

ORM-O was built within the scope of a project that aims to create a network of ontologies about software development frameworks. Such ontologies will allow us to automate semantic interoperability tasks, such as migrating code between frameworks, or specifying smells in software architecture independently of framework or language, for example.

As a proof of concept, we also developed a migration tool that converts code from one ORM framework to another (on different platforms) using ORM-O as an interlingua.

Keywords: Ontologies; Frameworks; Object/Relational Mapping; Object/Relactional Impedance Mismatch.

Lista de ilustrações

Figura 1 – Tipos de Ontologias pelo nível de generalidade segundo Guarino (1998). 18
Figura 2 – Tipos de Ontologias pelo nível de generalidade segundo Scherp et al. (2011). 18
Figura 3 – Fragmento do metamodelo do perfil UML da OntoUML (GUIZZARDI, 2005). 21
Figura 4 – Metamodelo da OntoUML (ONTOUML, 2019). 22
Figura 5 – Fases e processos do (FALBO, 2014) 23
Figura 6 – Arquitetura da RDBS-O. 24
Figura 7 – Diagrama OntoUML da DBS-O (AGUIAR; FALBO; SOUZA, 2018). . 24
Figura 8 – Diagrama OntoUML da RDBS-O (AGUIAR; FALBO; SOUZA, 2018). 25
Figura 9 – Composição da OOC-O. 26
Figura 10 – Diagrama OntoUML da OOC-O Core. 27
Figura 11 – Diagrama OntoUML da OOC-O Class. 27
Figura 12 – Diagrama OntoUML da OOC-O Class Member. 28
Figura 13 – Estrutura da OWL2, adaptada de (OWL Working Group, 2012). . . . 29
Figura 14 – Composição da ORM-O e sua associação com OOC-O e RDBS-O. . . . 34
Figura 15 – Subontologia ORM-O Class. 34
Figura 16 – Subontologia ORM-O Variable. 35
Figura 17 – Subontologia ORM-O Relationship. 36
Figura 18 – Relação entre Primary Key Column e Foreign Key Column. 37
Figura 19 – Relações derivadas por Mapped Foreign Key. 38
Figura 20 – Modelo OntoUML da subontologia ORM-O Inheritance. 39
Figura 21 – Modelo de classes do exemplo de domínio a ser instanciado pela ontologia. 43
Figura 22 – Modelo ER do banco de dados onde os objetos serão persistidos 44
Figura 23 – Fluxo de execução da ferramenta de migração. 48
Figura 24 – Parte da tela do Protégé com o carregamento da OOC-O e RDBS-O. . 48
Figura 25 – Árvore de hierarquia dos conceitos no Protégé. 49

Lista de tabelas

Tabela 1 – Exemplos de estereótipos OntoUML e suas distinções ontológicas – adaptada de (AGUIAR; FALBO; SOUZA, 2018) 19
Tabela 2 – Respostas às Questões de Competência. 41

Lista de abreviaturas e siglas

ANTLR	*Another Tool for Language Recognition*
API	*Application Programming Interface*
BD	Banco de Dados
CQ	*Competency Question*
DBS-O	*Database System Ontology*
JPA	Java Persistence API
OO	Orientação à Objeto
OOC-O	*Object-Oriented Code Ontology*
ORM-O	*Object/Relational Mapping Ontology*
ORM	*Objetct/Relational Mapping*
OWL	*Ontology Web Language*
RDBS-O	*Relational Database System Ontology*
RDF	*Resource Description Framework*
RNF	Requisito Não-funcional
SABiO	*Systematic Approach for Building Ontologies*
SEON	*Software Engineering Ontology Network*
SFWON	*Software Frameworks Ontology Network*
SPO	*Software Proccess Ontology*
SwO	*Software Ontology*
UFO	Unified Foundational Ontology
UML	*Unified Modeling Language*
XML	Extensible Markup Language

Sumário

1 INTRODUÇÃO 11
1.1 Objetivos e Método 12
1.2 Organização da Dissertação 13

2 REFERENCIAL TEÓRICO 14
2.1 Impedância Objeto/Relacional e Frameworks ORM 14
2.2 Ontologia 16
2.2.1 OntoUML 18
2.2.2 SABiO 20
2.3 Fundamentação Ontológica 22
2.3.1 RDBS-O 23
2.3.2 OOC-O 25
2.4 OWL2 28
2.5 Trabalhos Relacionados 30

3 PROPOSTA DO TRABALHO 32
3.1 ORM-O Class 34
3.2 ORM-O Variable 35
3.3 ORM-O Relationship 36
3.4 ORM-O Inheritance 38

4 AVALIAÇÃO DO TRABALHO 41
4.1 Verificação 41
4.2 Validação 43
4.3 Ferramenta de Migração 47

5 CONSIDERAÇÕES FINAIS 53

REFERÊNCIAS 55

1 Introdução

A evolução do desenvolvimento de software no passar dos anos trouxe diversas inovações, incluindo o surgimento de paradigmas, que são maneiras particulares de se ver um universo de discurso. Dentre os paradigmas que surgiram, dois deles se tornaram extremamente populares: o paradigma de orientação a objetos (OO) para o desenvolvimento de software e o paradigma relacional em banco de dados.

O uso conjunto desses dois paradigmas, um deles para escrita do programa e outro para persistência de dados, se tornou estado-da-prática e a diferença de abstração, foco e linguagem entre eles deu origem ao que chamamos de Impedância Objeto/Relacional (BAUER; KING, 2004).

Assim sendo, tornou-se estado da prática a delegação do mapeamento a *frameworks* de mapeamento objeto/relacional (*object/relational mapping*, ou ORM), devido ao benefício de reúso de código. Hoje, diferentes linguagens possuem diferentes *frameworks* ORM e, em muitos casos, até mesmo *frameworks* de uma mesma linguagem são diferentes sintaticamente quanto ao uso no código fonte.

Devido, então, a esse uso frequente de *frameworks* para o mapeamento objeto/relacional, buscou-se na literatura um padrão semântico que norteasse os conceitos implementados pelos diferentes *frameworks* nas diferentes linguagens.

A existência de uma definição semântica formal dos conceitos implementados pelos *frameworks* de mapeamento objeto/relacional é de grande valia, podendo ser usada para diversos fins, dentre eles: interpretação e entendimento dos *frameworks* atuais já propostos, base para desenvolvimento e proposta de novos *frameworks*, detecção de similaridades e equivalência entre diferentes *frameworks*, mesmo quando conceitos são abordados de formas diferentes, dentre outras.

Esse último uso pode ainda viabilizar a transcrição de códigos orientados a objeto com uso de um determinado *frameworks* para outros *frameworks* até mesmo de diferentes linguagens. Tal transcrição é útil, por exemplo, para unificação de programas, para usar uma única linguagem ou ainda para atualização de código quando a implementação que eles utilizam for descontinuada pelo desenvolvedor ou fornecedor.

Entretanto, a busca por tal padrão semântico não retornou resultados e então, levando isso em consideração, uma motivação para o desenvolvimento desse padrão surgiu. Tal padrão precisa levar em consideração a conceituação semântica que é compartilhada nos diferentes *frameworks*.

1.1 Objetivos e Método

Considerando o exposto e, ainda, levando em conta que ontologia é uma especificação explicita de uma conceituação (GRUBER, 1993), este trabalho tem como objetivo geral definir e formalizar uma representação ontológica do mapeamento objeto/relacional no escopo do código-fonte. Para tal, levamos em consideração que "representação ontológica" se refere ao conjunto de um modelo ontológico formal e sua implementação em uma linguagem capaz de ser lida por máquinas. Esse objetivo geral pode ser detalhado nos seguintes objetivos específicos:

- Formalizar um modelo ontológico que represente o domínio do mapeamento objeto/-relacional no escopo do código fonte;
- Avaliar o modelo construído, verificando que atenda seus requisitos e validando que represente bem o domínio em questão;
- Realizar uma prova de conceito com o uso da ontologia implementada em linguagem operacional em uma aplicação específica.

Para que esses objetivos da pesquisa fossem alcançados, uma revisão bibliográfica foi realizada com a finalidade de encontrar um referencial teórico adequado capaz de embasar a concepção de uma representação ontológica do nosso domínio. Com tal revisão, foram identificadas ontologias na literatura que puderam ser reutilizados (RDBS-O e OOC-O) e, para que o reúso fosse simplificado, utilizamos o mesmo método de Engenhariade Ontologias (SABiO), linguagem de modelagem gráfica (OntoUML) e linguagem operacional (OWL2).

Seguindo, uma vez que o propósito já tenha sido identificado, a apresentação de um modelo ontológico capaz de representar e formalizar o domínio do mapeamento objeto/relacional no escopo do código fonte só foi possível após a realização de um processo de aquisição de conhecimento de *frameworks* já existentes. Assim, utilizando OntoUML assim como nas ontlogias reutilizadas, o modelo foi construído.

Com o modelo ontológico construído, passou-se para atividades que compusessem uma avaliação. Para tal, a cobertura da ontologia foi verificada e também uma validação viabilizada através de exemplos de uso real. Modelagem e avaliação foram conduzidas de forma iterativa, de modo que o modelo passou por diversas evoluções à medida que era avaliado.

Por fim, uma prova de conceito só foi possível após a implementação da ontologia em uma linguagem operacional. Após isso, uma ferramenta de migração de código ORM entre diferentes linguagens/*frameworks* foi desenvolvida.

1.2 Organização da Dissertação

Nesse primeiro capítulo da dissertação foram apresentadas as ideias gerais deste trabalho, foram descritos o contexto de aplicação, os objetivos e a metodologia adotada. Além dessa introdução este documento organiza-se da seguinte forma:

- Capítulo 2 – Fundamentação Teórica: apresenta aspectos teóricos relacionados a ontologias, representação ontológica na linguagem OntoUML e desenvolvimento de ontologias com o método SABiO. Além disso, introduz as ontologias de código orientado a objetos e de banco de dados relacionais (OOC-O e RDBS-O), cujos conceitos são reutilizados por esse trabalho, bem como a linguagem de implementação de ontologias operacionais por nós utilizada (OWL2). Por fim, apresenta o problema da impedância objeto/relacional e como os *frameworks* de mapeamento podem auxiliar no desenvolvimento de sistemas quanto a isso;
- Capítulo 3 – Proposta: apresenta a ORM-O, o modelo ontológico de mapeamento objeto/relacional, que é a proposta deste trabalho;
- Capítulo 4 – Avaliação: descreve os resultados das atividades de verificação e validação, demonstrando a cobertura da ontologia no domínio do mapeamento objeto/relacional. Também apresenta a prova de conceito com o uso da ontologia, implementada operacionalmente, em uma ferramenta de migração de código ORM entre diferentes linguagens/*frameworks*;
- Capítulo 5 – Considerações Finais: apresenta as conclusões do trabalho, as contribuições e propostas de trabalhos futuros para continuidade e aprimoramento das contribuições aqui descritas.

O trabalho descrito nesta dissertação foi publicado no artigo "Representação Ontológica de Frameworks de Mapeamento Objeto/Relacional" (ZANETTI; AGUIAR; SOUZA, 2019), apresentado no 12º Seminário de Pesquisa em Ontologia no Brasil.

2 Referencial Teórico

Neste capítulo serão apresentados conceitos no estado da arte de assuntos que dão o embasamento teórico para a criação e desenvolvimento desse trabalho. Também será apresentado o problema da impedância objeto/relacional e a solução que os *frameworks* de mapeamento entre os dois paradigmas traz para os desenvolvedores de sistemas.

O capítulo é divido da seguinte forma: a Seção 2.1 faz uma apresentação sobre o problema da impedância objeto/relacional e a forma de solução utilizando *frameworks* ORM; a Seção 2.2 apresenta uma introdução sobre ontologia, incluindo breves descrições sobre a linguagem OntoUML (Seção 2.2.1) e o método de Engenharia de Ontologias SABiO (Seção 2.2.2); a Seção 2.3 trata sobre as representações ontológicas de bancos de dados relacionais e de código orientado a objetos, reutilizadas neste trabalho; por fim, a Seção 2.4 traz um panorama sobre a OWL2, uma linguagem para implementação de ontologias operacionais.

2.1 Impedância Objeto/Relacional e Frameworks ORM

Com a evolução no desenvolvimento de software no decorrer do tempo, dois diferentes paradigmas foram firmados como estado-da-prática: o paradigma de objetos e o relacional.

Um paradigma é uma maneira particular de ver um universo de discurso. Cada paradigma vem com suas próprias abstrações particulares, organizando princípios e preconceitos. Existem vários paradigmas diferentes na computação. Cada paradigma influencia o processo e os artefatos de design e desenvolvimento de software (IRELAND et al., 2009).

O paradigma relacional em banco de dados foi proposto por Codd (1970) e usa um conceito de relação matemática como estrutura básica, tendo sua base a teoria de conjuntos e a lógica de predicados de primeira ordem. Nessa abordagem as informações são persistidas exclusivamente em tabelas, de forma que possam ser acessadas logicamente, recorrendo a uma combinação de nome da tabela, valor da chave primária e nome da coluna.

Já no paradigma de objetos, as informações das instâncias de uma classe são armazenadas em memória como um objeto que tem características, atributos e comportamentos.

Desde suas respectivas propostas, o paradigma relacional provou ser popular para o desenvolvimento de bancos de dados, ao mesmo tempo em que o paradigma de objetos sustentou várias linguagens de programação e métodos de desenvolvimento de software. A popularidade das tecnologias que incorporam os diferentes paradigmas nesses dois aspectos

separados, porém essenciais, do desenvolvimento de software significa que inevitavelmente eles serão usados em conjunto.

Uma aplicação objeto/relacional combina artefatos de ambos paradigmas: linguagens orientadas a objetos são utilizadas na escrita do programa enquanto bancos de dados relacionais são utilizados no armazenamento das informações. Porém, diferenças de abstração, foco e linguagem levam a problemas quando essas tecnologias são combinadas em um único aplicativo (IRELAND et al., 2009). Esse problema na diferença dos paradigmas é chamado de Impedância Objeto/Relacional (*object/relational impedance missmatch*) (BAUER; KING, 2004).

Dessa maneira, para que o problema seja contornado, o desenvolvedor deve realizar o mapeamento de um objeto em memória para o banco de dados e, de maneira oposta, realizar a consulta no banco de dados relacional para recuperar as informações do objeto e então poder instanciá-lo novamente. Segundo Bauer e King (2004), o mapeamento objeto/relacional é a persistência automatizada e transparente de objetos de uma aplicação escrita em uma linguagem orientada a objetos para as tabelas de um banco de dados relacional, utilizando metadados que descrevem o mapemento entre os objetos e o banco de dados. Em essência, funciona transformando dados de uma representação a outra.

Como o passar do tempo, esse mapeamento passou a ser delegado a um *framework*, que é um conjunto de códigos que fornece solução para algum problema específico. O uso de *frameworks* se tornou estado-da-prática, pois reduz consideravelmente o tempo de desenvolvimento de um projeto por reutilizar código já desenvolvido, testado e documentado por terceiros (SOUZA; FALBO; GUIZZARDI, 2009).

O objetivo é que, ao invés de fazer manualmente o mapeamento objeto/relacional, o desenvolvedor utilize um *framework* ORM (*Object/Relational Mapping*) que, a partir de determinadas configurações, se encarrega do mapeamento dos objetos para tabelas e colunas ou a recuperação dos objetos por meio de consultas. Para tal, um *framework* ORM precisa prover (BAUER; KING, 2004):

- Uma API (métodos que podem ser chamados) para execução de operações básicas de persistência (denominadas CRUD, acrônimo para *create, retrieve, update and delete*, ou inserir, recuperar, atualizar e excluir, em português);
- Uma linguagem ou API para a construção de consultas orientadas a objetos, i.e., que se referem às classes e suas propriedades ao invés de tabelas e colunas, que são transformadas em consultas relacionais para recuperação de dados no banco de dados;
- Especificação dos metadados de mapeamento, que são utilizados pelo desenvolvedor para indicar como transformar os dados de uma representação a outra, fundamental

para o funcionamento de todo o *framework*;

- Implementação de técnicas de interação com o banco de dados relacional, de modo que o mapeamento objeto/relacional seja o mais transparente e otimizado possível.

Dada a popularidade deste tipo de solução, existem diferentes *frameworks* ORM para diversas linguagens de programação OO como, por exemplo: Hibernate (Java),[1] EclipseLink (Java),[2] Django (Python),[3] SQLAlchemy (Python),[4] ODB (C++),[5] QxOrm (C++),[6] dentre outros.

Para o uso destes *frameworks*, são necessários, além de inclusão de bibliotecas, pacotes e arquivos de configuração, adaptações ao código desenvolvido. Os *frameworks* de Java obedecem ao padrão JPA (Java Persistence API)[7] e utilizam trechos de código precedidos de `@` como anotações para indicação do comportamento de classes e atributos em relação aos *frameworks*. Em Python os *frameworks* possuem classes a serem estendidas ou instanciadas e métodos a serem utilizados. Em C++ são utilizados a diretiva `pragma` e os *templates* das bibliotecas dos *frameworks*.

2.2 Ontologia

O termo ontologia vem do século 17, paralelamente nas obras *Lexicon philosophicum* e *Ogdoas Scholastica* dos filósofos Rudolf Göckel e Jacob Lorhard, respectivamente. Porém, o termo ganhou grande popularidade somente no século 18 através da publicação do *Philosophia prima sive Ontologia* de Christian Wolff (GUIZZARDI, 2005). Em seu trabalho, Guizzardi (2005) comenta que etimologicamente *ont-* vem do verbo grego *einai* (ser), portanto a palavra latina Ontologia (ont + logia) pode ser traduzida como o estudo da existência.

Diversos estudos e definições a respeito de ontologia são encontrados na literatura. No trabalho de Gruber (1993) é dito que ontologia é uma especificação explícita de uma conceituação, onde especificação é no sentido de representação formal e declarativa e conceituação se refere a uma abstração, uma visão simplificada do mundo. Segundo Guarino (1995), ontologia é o estudo da organização e da natureza do mundo.

Hendler (2001) utiliza o termo como um conjunto de termos de conhecimento e algumas regras simples de inferência lógica de um assunto específico. Já em (SWARTOUT;

[1] <https://hibernate.org>
[2] <https://www.eclipse.org/eclipselink/>
[3] <https://www.djangoproject.com>
[4] <https://www.sqlalchemy.org>
[5] <http://www.codesynthesis.com/products/odb>
[6] <https://www.qxorm.com>
[7] <https://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html>

TATE, 1999) os autores consideram que uma ontologia fornece a estrutura ou armadura básica em torno da qual uma base de conhecimento pode ser construída. Para eles uma ontologia fornece um conjunto de conceitos e termos para descrever algum domínio, enquanto uma base de conhecimento usa esses termos para representar o que é verdadeiro sobre algum mundo real ou hipotético.

A palavra Ontologia tende a ser utilizada de diferentes maneiras e, por isso, seu significado tende a permanecer um pouco vago (GIARETTA; GUARINO, 1995). Guarino, Oberle e Staab (2009) dizem que o uso da palavra é distinguido quando usado com letra inicial maiúscula ou minúscula. Segundo estes autores, "Ontologia" remete a uma disciplina do ramo da filosofia que trata da natureza e estrutura da realidade e que, segundo Aristóteles, é o estudo de atributos que pertencem às coisas devido a seu estado natural. Já "ontologia" é utilizado em Ciência da Computação e se refere a um tipo especial de objeto de informação ou artefato computacional e, nesse caso, o relato da existência é pragmático onde o que existe é aquilo que pode ser representado.

Além das diferentes definições para o termo, também existem diferentes tipos de ontologias (GUARINO, 1997). O primeiro tipo diz respeito ao formalismo da expressividade das relações e dos conceitos, nesse contexto existem "ontologias leves" e "ontologias pesadas" (*lightweight ontologies* e *heavyweight ontologies*, respectivamente), sendo que a primeira define a taxonomia que representa a relação hierárquia entre conceitos sem se ocupar detalhadamente com esses conceitos. Por outro lado, "ontologias pesadas", além da taxonomia, define cada conceito e sua organização por meio de princípios que estejam bem definidos, gerando uma representação mais rigorosa da semântica entre os conceitos (ISOTANI; BITTENCOURT, 2015).

Já de acordo com o nível de generalidade, podem existir ontologias de fundamentação, de domínio, de tarefa ou ainda de aplicação (GUARINO, 1998). As consideradas *Ontologias de Fundamentação* são aquelas mais abrangentes, que não dependem de um domínio particular e descrevem conceitos gerais. A partir do momento que uma ontologia de fundamentação é especializada e utiliza termos específicos de um domínio e passa a representar seus conceitos, ela é dita como *Ontologia de Domínio*.

Seguindo a ideia, quando uma ontologia de fundamentação é especializada para definir conceitos sobre resoluções de problemas através da conceituação de uma atividade ou tarefa genérica, a ontologia é considerada como *Ontologia de Tarefa*. Nesse contexto, por fim, uma tarefa de um domínio específico dá origem às *Ontologias de Aplicação* (GUARINO, 1998; CAMPOS et al., 2012). A Figura 1 ilustra de forma gráfica as relações entre os tipos de ontologia pela generalidade.

Ainda pelo nível de generalidade, Scherp et al. (2011) fez uma proposta parecida com a de Guarino (1998), considerando ontologias de fundamentação e de domínio, porém adicionando um nível entre elas, as chamadas *Ontologias de Núcleo*. Essas ontologias,

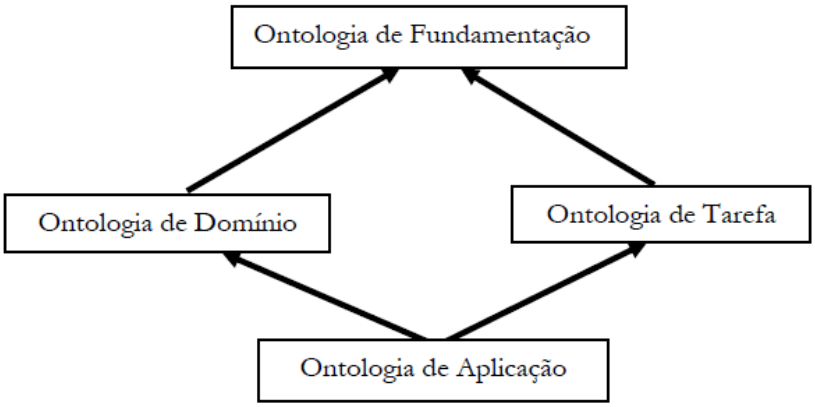

Figura 1 – Tipos de Ontologias pelo nível de generalidade segundo Guarino (1998).

Figura 2 – Tipos de Ontologias pelo nível de generalidade segundo Scherp et al. (2011).

segundo o autor, definem uma definição abstrata detalhada de um conhecimento estruturado de uma área definida, por exemplo, Medicina, Direito, serviços de software, etc., e podem ser baseadas em ontologias de fundamentação através de um refinamento por adição de relações e conceitos detalhados da área específica. A hierarquia por ele porposta está representada na Figura 2.

Também há uma relevante distinção entre *Ontologias de Referência* e *Ontologias Operacionais*. Nesse contexto, uma ontologia de referência é feita objetivando descrever da melhor forma algum domínio da realidade, sendo assim um tipo especial de modelo conceitual. Desta forma, uma vez que a conceituação comum tenha sido aceita, uma versão operacional é desenvolvida com a finalidade de garantir propriedades computacionais desejadas (GUIZZARDI, 2007).

2.2.1 OntoUML

Guizzardi (2005) destacou o status da UML como linguagem de modelagem padrão *de facto* e o crescente interesse em sua adoção como linguagem para modelagem conceitual. Porém, como uma linguagem de representação de modelagem conceitual de ontologia, o autor relata que o metamodelo da UML pode conter casos de incompletude, sobrecarga,

redundância e excesso de construtos.

Diante disso, houve a proposta de uma série de modificações no metamodelo da UML, produzindo uma versão ontologicamente bem fundamentada e de semântica formal, além de um adicional suporte metodológico ao usuário da linguagem na decisão de como modelar (GUIZZARDI, 2005). Tal proposta foi realizada com base na UFO (*Unified Foundational Ontology*) (GUIZZARDI, 2005), que é uma ontologia de fundamentação embasada em uma série de teorias de áreas como Linguística, Psicologia Cognitiva, Ontologias Formais e Lógica Filosófica, provendo uma representação sólida da conceituação de entidades e suas relações existentes no mundo real.

Sendo assim, essas modificações geraram um perfil UML — a OntoUML — e um conjunto de estereótipos representando as categorias ontológicas dos tipos universais propostos em UFO, bem como restrições formais que refletem a axiomatização de UFO, fazendo com que modelos válidos em OntoUML sejam restritos a representações compatíveis com UFO (GUIZZARDI, 2007).

Na Figura 3 é possível ver um fragmento da extensão do metamodelo da UML que gerou o perfil correspondente da OntoUML. Essa extensão da UML é baseada no metamodelo da OntoUML, representado na Figura 4. Por sua vez, a Tabela 1 apresenta alguns exemplos do que os estereótipos de OntoUML representam.

Tabela 1 – Exemplos de estereótipos OntoUML e suas distinções ontológicas – adaptada de (AGUIAR; FALBO; SOUZA, 2018)

Estereótipo	**Distinção Ontológica**	**Exemplo**
`<<category>>`	Um conceito cujas instâncias compartilham propriedades comuns, mas obedecem a diferentes princípios de identidade.	Cliente
`<<collective>>`	Um complexo funcional formado por partes iguais, com o princípio de identidade do proprietário, mantido tanto por suas instâncias quanto em todos os mundos possíveis.	Delegação
`<<kind>>`	Um complexo funcional que representa conceitos rígidos que fornecem um princípio de identidade para suas instâncias e não exigem uma dependência relacional.	Pessoa
`<<mode>>`	Um conceito cujas instâncias representam propriedades intrínsecas de um indivíduo.	Dor de Cabeça
`<<relator>>`	Uma dependência relacional entre conceitos.	Casamento
`<<role>>`	Um conceito anti-rígido, isso é, cujo as instâncias não possuem o mesmo princípio de identidade, tornando-se elacionalmente dependente de um princípio rígido de identidade.	Estudante

continua na próxima página

Tabela 1. Continuação da página anterior

Estereótipo	**Distinção Ontológica**	**Exemplo**
`<<subkind>>`	Um conceito cujas instâncias herdam um princípio de identidade de um `kind`, tornando-se uma especialização rígida.	Mulher

2.2.2 SABiO

O desenvolvimento de ontologias é feito através de etapas de métodos de Engenharia de Ontologias. Em especial, uma abordagem baseada em processos de Engenharia de Software é o SABiO (FALBO, 2014) (*Systematic Approach to Build Ontologies*). O SABiO compreende fases e processos que estão representados na Figura 5.

Como pode ser observado na Figura 5, SABiO é composto por cinco fases que serão brevemente descritas a seguir:

- **Identificação do Propósito e Elicitação de Requisitos**: nesta primeira fase, são identificados o propósito da ontologia e de que maneira ela será utilizada. Aqui ocorrem as identificações dos requisitos que são descritos na forma de Questões de Competências (CQ – *Competency Questions*). Tais questões devem ser respondidas pela ontologia. Também são elicitados os requisitos não-funcionais, que são aspectos gerais não relacionados ao domínio;

- **Captura e Formalização da Ontologia**: aqui, com base no propósito e nos requisitos levantados na fase anterior, ocorre a conceituação da ontologia. Assim, conceitos, relações e propriedades relevantes devem ser identificados e organizados. São produzidos modelos usando linguagem gráfica com a finalidade de ser utilizada por humanos. Após essa segunda fase, o principal resultado obtido é a ontologia de referência;

- **Projeto**: fase em que ocorre a transformação da especificação conceitual da ontologia de referência em uma especificação de projeto. Para isso devem ser levadas em consideração uma série de questões que variam, desde ambientes arquiteturais e requisitos tecnológicos não-funcionais até um ambiente de implementação específico;

- **Implementação**: trata da codificação da ontologia em uma linguagem operacional previamente escolhida na fase anterior. Ao fim da fase de implementação tem-se como resultado uma versão operacional da ontologia de referência;

- **Teste**: nessa última fase, ocorre a verificação e validação da ontologia operacional utilizando um conjunto de casos de teste. Os testes são então comparados com o que é esperado em relação às questões de competência.

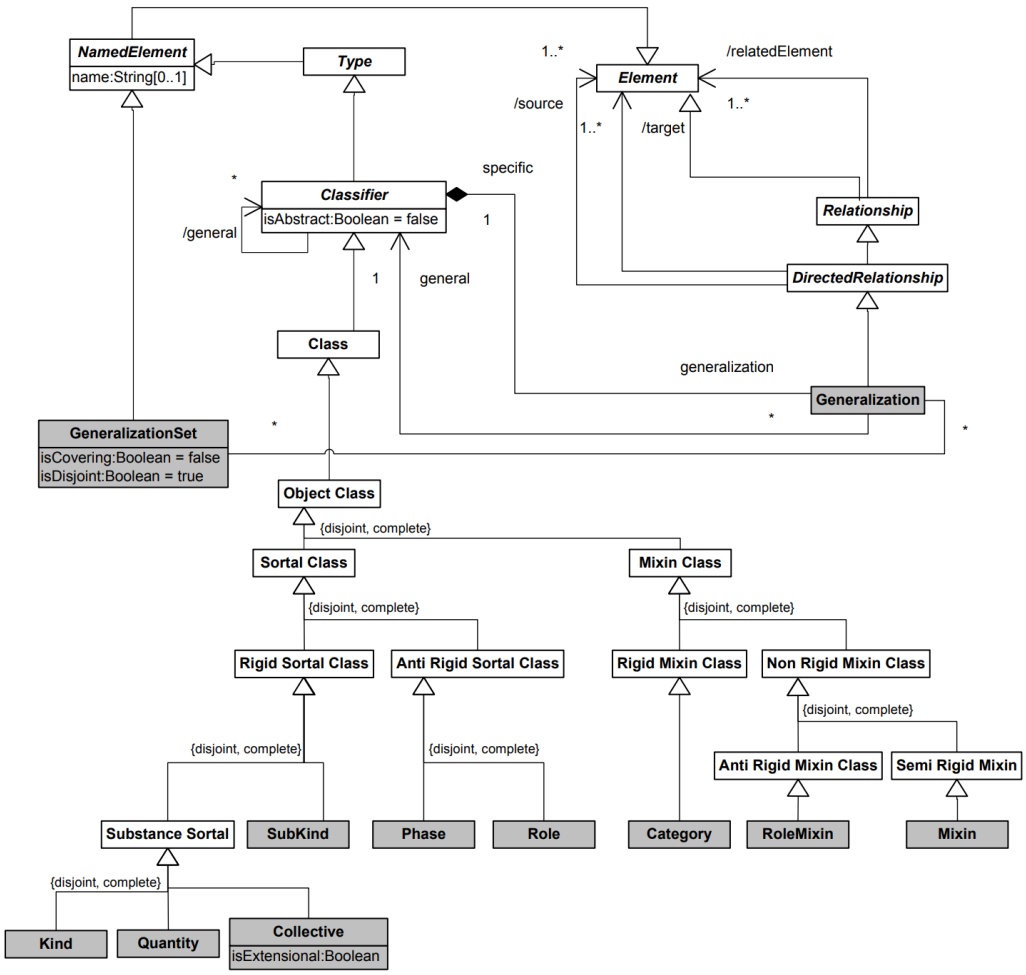

Figura 3 – Fragmento do metamodelo do perfil UML da OntoUML (GUIZZARDI, 2005).

Em paralelo às fases que dizem respeito ao processo de desenvolvimento, existem alguns processos de apoio indicados pelo método:

- **Aquisição de Conhecimento**: ocorre principalmente nas fases iniciais do processo de desenvolvimento da ontologia. Esse processo trata da aplicação de métodos e técnicas convencionais para aquisição de conhecimento e elicitação de requisitos, principalmente aqueles dedicados à aquisição de conhecimento colaborativo, como o *brainstorming*;
- **Reúso**: existem muitas oportunidades para reutilizar conceituações já estabelecidas para o domínio. As principais fontes de reutilização são: ontologias de domínio existentes, ontologias principais, ontologias fundamentais e padrões de ontologias;
- **Documentação**: todo o processo de desenvolvimento deve ser documentado, incluindo os resultados de alguns processos de suporte, como a avaliação;
- **Gerência de Configuração**: trata da gerência da configuração dos principais documentos propostos pelo SABiO;

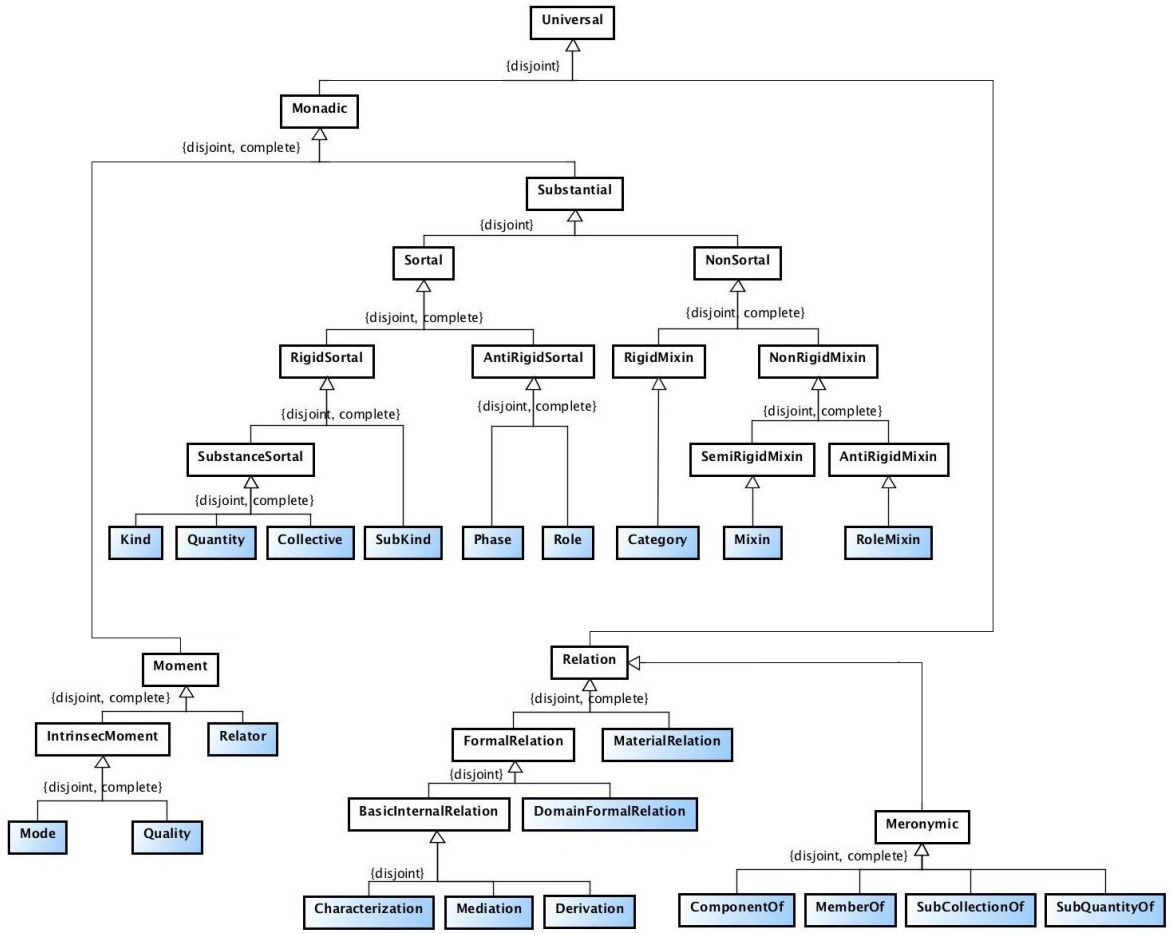

Figura 4 – Metamodelo da OntoUML (ONTOUML, 2019).

- **Avaliação**: a fase de Teste, descrita no processo de desenvolvimento é, de fato, uma atividade de avaliação, porém uma avaliação dinâmica. Por outro lado há várias outras atividades de avaliação estáticas, como a revisão técnica, que são necessárias e devem ser executadas durante o processo de desenvolvimento da ontologia com o intuito de avaliar o trabalho intermediário do processo de desenvolvimento e determinar sua adequação ao uso pretendido.

2.3 Fundamentação Ontológica

Para a construção de uma ontologia sobre mapeamento objeto/relacional, reutilizamos ontologias existentes sobre ambos os paradigmas. Tais ontologias fazem parte de uma A Seção 2.3.1 descreve a ontologia de sistemas de bancos de dados relacionais (*Relational Database System Ontology* – RDBS-O), enquanto a Seção 2.3.2 descreve a ontologia de código orientado a objetos (*Object-Oriented Code Ontology* – OOC-O).

É importante ressaltar ainda que todas elas fazem reúso de uma coleção de ontologias relacionadas entre si através de alinhamento, modularização ou dependências — o que

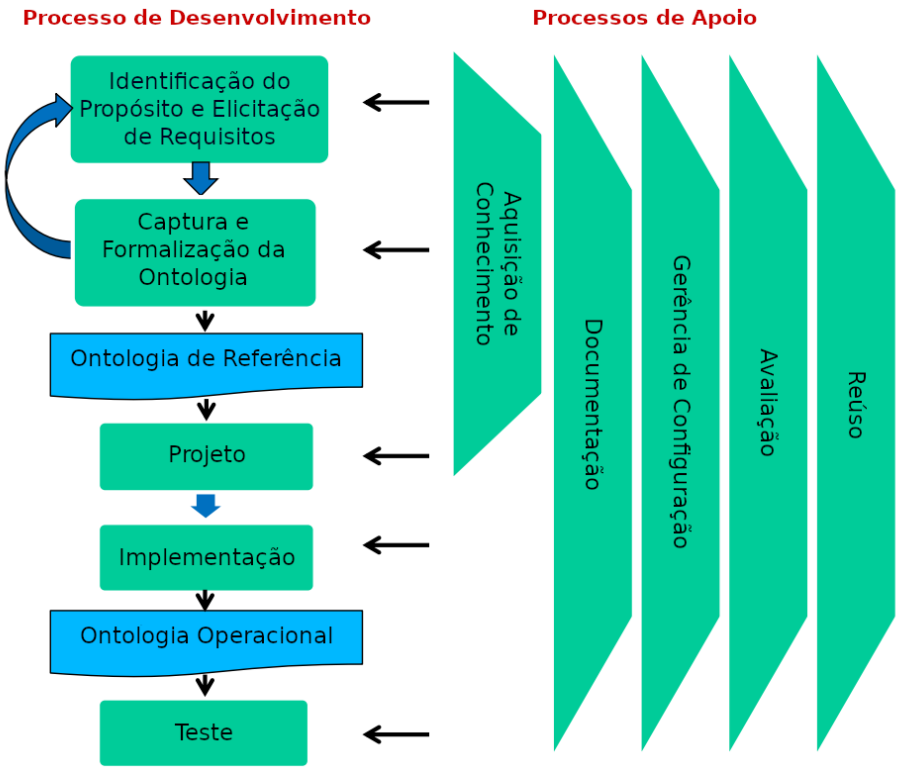

Figura 5 – Fases e processos do (FALBO, 2014)

chamamos de Rede de Ontologias (DAQUIN; GANGEMI; HAASE, 2006).

2.3.1 RDBS-O

A ontologia de Sistemas de Banco de Dados Relacionais (*Relational Database System Ontology* – RDBS-O) foi proposta por Aguiar, Falbo e Souza (2018) para representar os principais conceitos pertinentes ao domínio de banco de dados no escopo arquitetural, descartando detalhes de controle e execução.

A RDBS-O reutiliza duas ontologias propostas na Rede de Ontologias de Engenharia de Software (*Software Engineering Ontology Network* – SEON) (RUY et al., 2016): a Ontologia de Processo de Software (*Software Process Ontology* – SPO) (BRINGUENTE; FALBO; GUIZZARDI, 2011), que estabelece uma conceituação comum no domínio de processo de software, incluindo processos atividades, recursos, pessoas, artefatos e procedimentos e a Ontologia de Software (*Software Ontology* – SwO) (DUARTE et al., 2018), que captura os produtos de software que são constituídos por artefatos de software de diferentes naturezas, incluindo sistemas de software, arquivo de dados, programas e código.

Além disso, Aguiar, Falbo e Souza (2018) propõem um módulo da ontologia que representa sistemas de banco de dados em geral, chamado DBS-O *(Database Systems Ontology*). Dessa maneira uma arquitetura é composta e pode ser observada na Figura 6.

Após a identificação de questões de competência e uma sessão de aquisição de

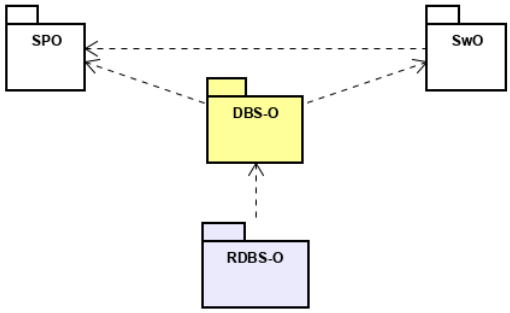

Figura 6 – Arquitetura da RDBS-O.

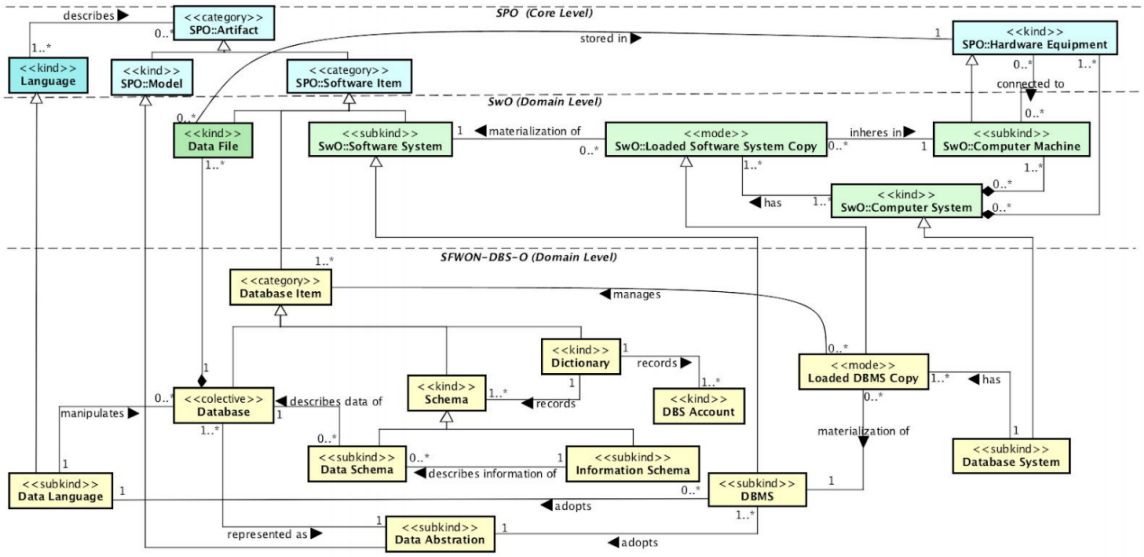

Figura 7 – Diagrama OntoUML da DBS-O (AGUIAR; FALBO; SOUZA, 2018).

conhecimento, um diagrama é apresentado em OntoUML e está reproduzido na Figura 7, que é a representação ontológica da DBS-O. Nessa representação é possível notar a definição de alguns conceitos, como o **Database Item** e suas especializações, que são gerenciadas por um **Loaded DBMS Copy**. Um **Database** é manipulado por um **Data Language**, cujo as instâncias são linguagens declarativas que especificam o que deve ser feito para manipular dados. Um **Dictionary** descreve **DBS Accounts** e **Schemas**, que por usa vez descrevem um **Database**.

Uma vez que a DBS-O foi definida, a RDBS-O pôde, então, começar a ser desenvolvida. Mais uma vez questões de competência foram identificadas e descritas e então um modelo ontológico em OntoUML apresentado, como na sua reprodução visível na Figura 8. Nela é possível notar que um **Relational Database System** é composto por **RDBMS Items**, que representam os dados de acordo com o modelo relacional através de **Tables** e definem restrições sobre o modelo na forma de **Constraints**.

Também é possível notar que **Table** é definido por **Line Type**, que por sua vez é constituído de **Columns**, representando os campos da tabela. Uma **Column** é especificada

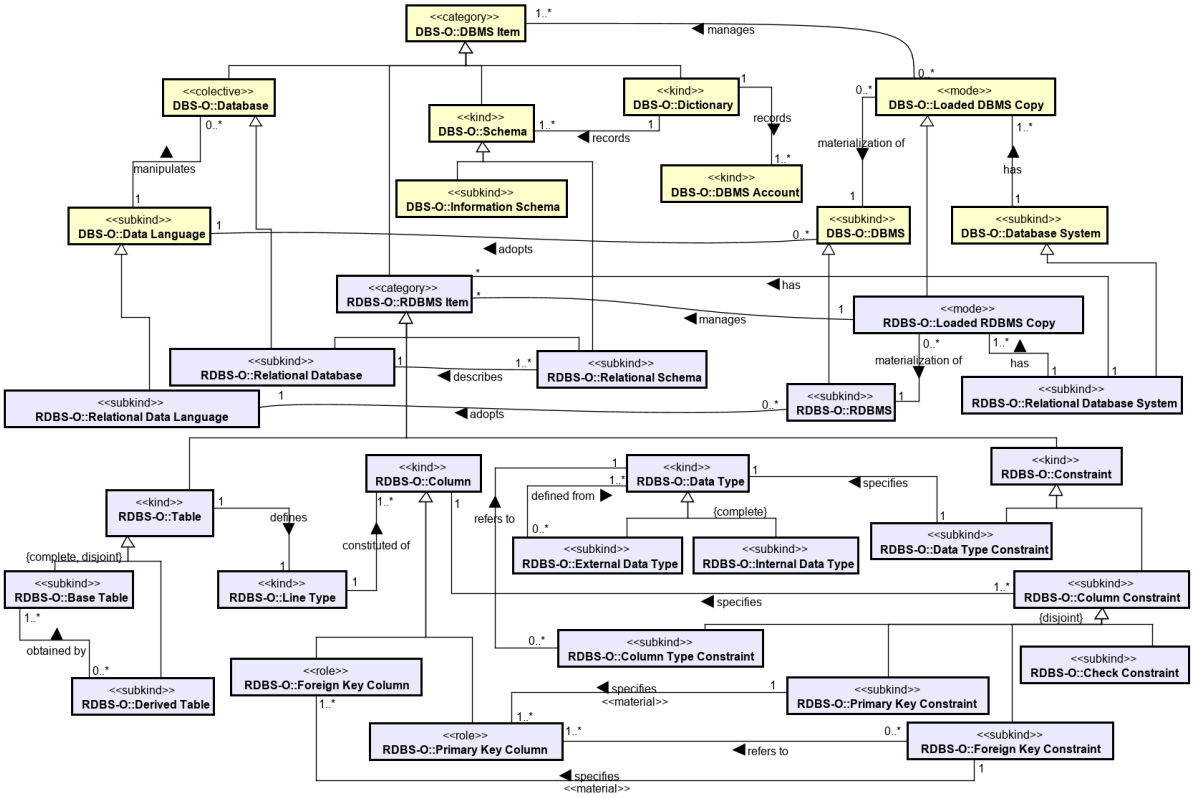

Figura 8 – Diagrama OntoUML da RDBS-O (AGUIAR; FALBO; SOUZA, 2018).

por **Column Constraint**, estabelecendo, assim, restrições.

O documento de especificação de ontologia que contém a descrição completa da DBS-O e RDBS-O, bem como as questões de competência respondidas pelas ontologias, dicionário de termos e uma avaliação composta por etapas de verificação e validação está disponível na páginas do projeto SFWON.[8]

No contexto deste trabalho, que representa o domínio dos *frameworks* ORM, os conceitos de **Table** e **Column** são essenciais, visto que o mapeamento associa classes e atributos das linguagens orientadas a objetos respectivamente a estes conceitos do modelo relacional. Em particular, os conceitos de **Primary Key Column** e **Foreign Key Column** permitem a representação do mapeamento de atributos de identificação única de objetos e de associações entre classes, respectivamente.

2.3.2 OOC-O

A Ontologia de Código Orientado a Objetos (*Object-Oriented Code Ontology* – OOC-O) (AGUIAR; FALBO; SOUZA, 2019) identifica e representa a semântica das entidades presentes em tempo de compilação no código orientado a objetos (OO). Em tempo de compilação implica dizer que, embora os objetos sejam as construções fundamentais na

[8] *Software Frameworks Ontology Network*: <https://nemo.inf.ufes.br/projects/sfwon/>

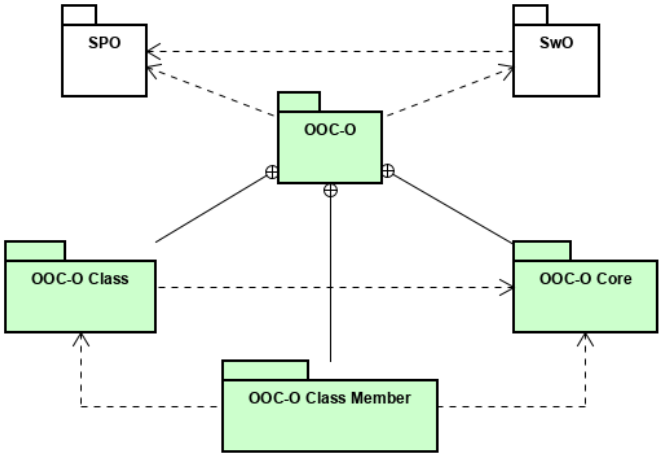

Figura 9 – Composição da OOC-O.

programação OO, eles existem apenas em tempo de execução e, portanto, não são cobertos pelo OOC-O.

A OOC-O é composta de três módulos: **OOC-O Core**, que contém uma visão geral dos princípios e conceitos, **OOC-O Class**, que detalha conceitos derivados de Classe, e **OOC-O Class Member**, que detalha conceitos derivados de membros de Classe, ou seja, métodos e variáveis.

Além disso, a OOC-O também faz uso das ontologias SPO (BRINGUENTE; FALBO; GUIZZARDI, 2011) e SwO (DUARTE et al., 2018), ambas fazem parte da SEON (RUY et al., 2016) já brevemente introduzidas na Seção 2.3.1. Sendo assim, a arquitetura da OOC-O é a que está representada na Figura 9.

A Figura 10 mostra os conceitos da **OOC-O Core** e como são integrados com as ontologias SPO e SwO de SEON. Nela são introduzidos alguns conceitos, como **Object-Oriented Source Code** que é representado por uma **Object-Oriented Programming Language** e constituído de **Physical Module**, que por sua vez são compostos de **Classes** que estão organizadas em **Logical Modules**.

Classes são compostas de **Members**, sejam eles **Methods** ou **Attributes**.Todos esses são **Named Elements**, caracterizados por **Name** e **Visibility**.

A Figura 11 mostra os conceitos da **OOC-O Class**. Notamos ali que cada **Class** deve ser **Concrete Class** ou **Abstract Class**. Uma **Abstract Class** é, obrigatoriamente, uma **Extendable Class** que pode assumir um papel de **Superclass** uma vez que tenha uma relação nomeada **Inheritance** com uma **Subclass** que é outro possível papel de **Class**.

Por fim, a Figura 12 mostra o diagrama OntoUML da OOC-O **Class Member**. Nota-se que todo **Method** deve ser um **Concrete Method** ou um **Abstract Method**. O primeiro pode ser especializado em **Class Method** ou **Instance Method**. Um **Instance**

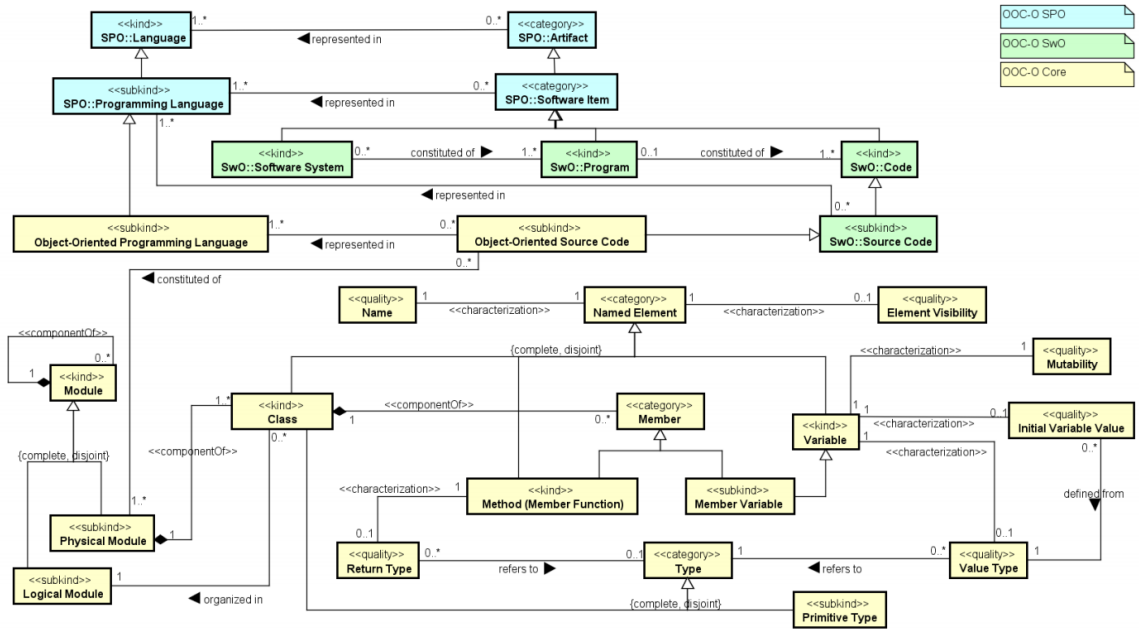

Figura 10 – Diagrama OntoUML da OOC-O Core.

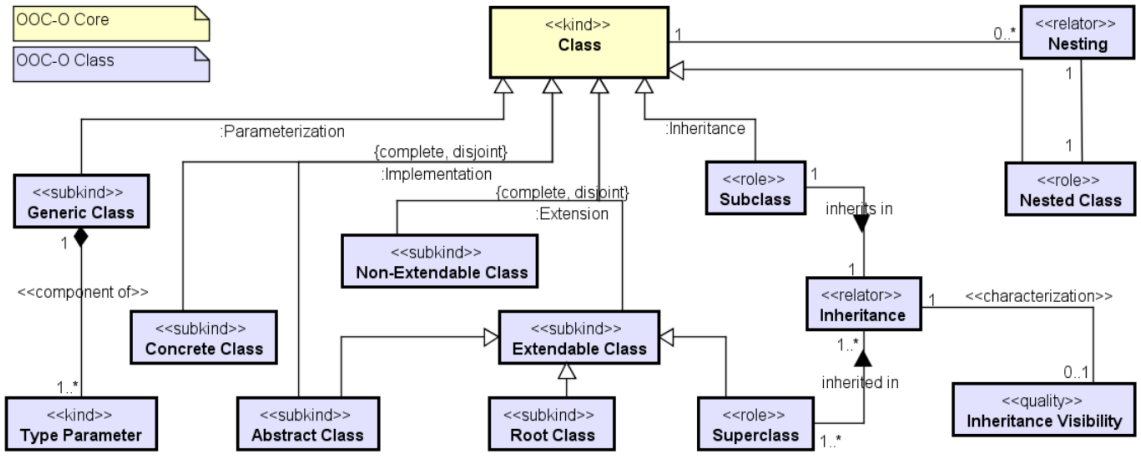

Figura 11 – Diagrama OntoUML da OOC-O Class.

Method pode ser especializado em **Constructor Method**, **Destructor Method** ou **Accessor Method**. Além disso, **Attributes** podem ser **Instance Variable** ou **Class Variable** e **Method Variable** pode ser **Parameter Variable** ou **Local Variable**

O documento de especificação de ontologia que contém a descrição completa da OOC-O e contém as questões de competência respondidas pelas ontologias, dicionário de termos e uma avaliação composta por etapas de verificação e validação também está disponível na página do projeto SFWON.

Para o desenvolvimento deste trabalho, conceitos representados pela OOC-O são fundamentais e por isso são especializados. O conceito **Class** e **Instance Variable** se destacam visto que representam classes e atributos em uma linguagem orientada a objeto

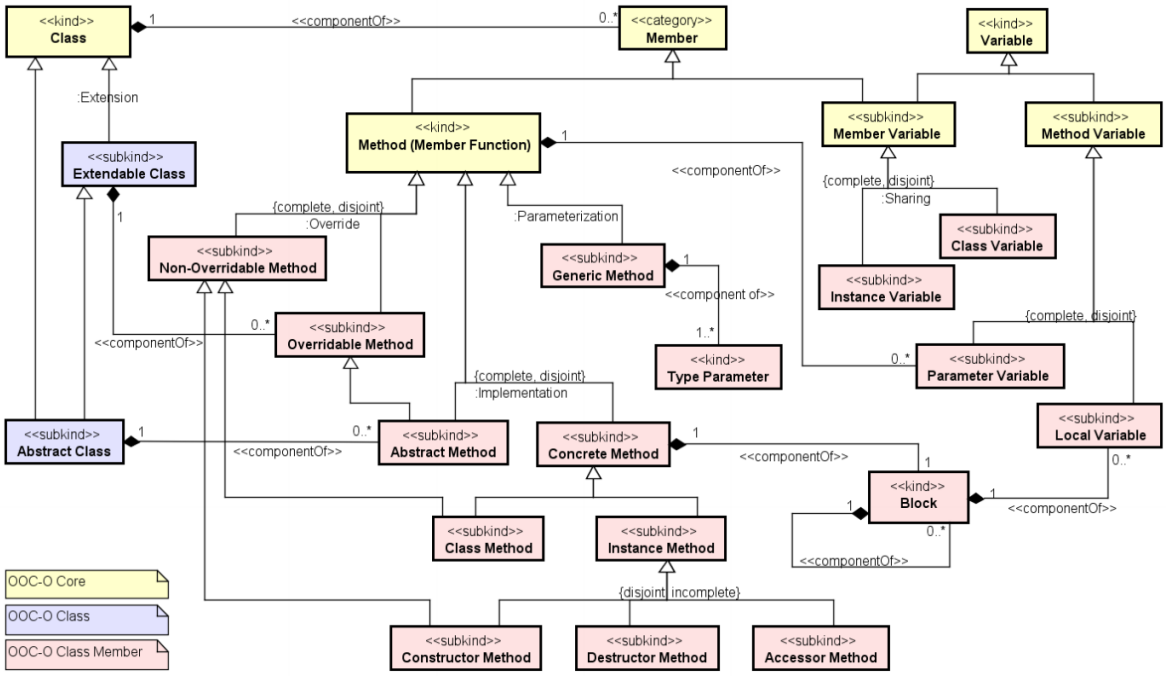

Figura 12 – Diagrama OntoUML da OOC-O Class Member.

e são mapeadas para o modelo relacional. Outros conceitos como **Superclass** e **Subclass**, apesar de serem especializações do conceito **Class**, também são importantes pois implicam em diferentes tipos de mapeamento ao banco de dados relacional.

2.4 OWL2

A OWL2 (*Web Ontology Language*) (OWL Working Group, 2012) é uma linguagem operacional de representação de ontologias voltada para a Web Semântica, com significado formalmente definido. Ontologias implementadas em OWL2 fornecem classes, propriedades, indivíduos e valores de dados e são armazenadas como documentos da Web Semântica. A Figura 13 fornece uma visão geral da linguagem OWL2, mostrando seus principais componentes e como eles se relacionam.

A elipse no centro representa a noção abstrata de uma ontologia, que pode ser representada de duas maneiras. A primeira é através de uma estrutura conceitual, que em OWL2 tem sua definição no documento de especificação estrutural (MOTIK; PATEL-SHNEIDER; PARSIA, 2012). Tal documento é baseado em UML (Object Management Group, 2007) e suas definições. Outra maneira é a representação através de grafo RDF que é gerada por meio de um mapeamento da representação conceitual que tem sua definição em um documento de mapeamento para RDF (PATEL-SHNEIDER; MOTIK, 2012).

Na parte superior estão várias sintaxes concretas que podem ser usadas para serializar. A principal sintaxe é a RDF/XML (BACKETT, 2014) que, segundo o documento de conformidade da OWL2 (HAWKDE et al., 2012), é a única sintaxe obri-

Figura 13 – Estrutura da OWL2, adaptada de (OWL Working Group, 2012).

gatoriamente suportada por todas as ferramentas da OWL2. Alternativamente outras sintaxes podem ser utilizadas para serializações RDF, como a Turtle (BACKETT et al., 2014), OWL/XML (MOTIK; PARSIA; PATEL-SHNEIDER, 2012) e Manchester (HORRIDGE; PATEL-SHNEIDER, 2012), além de uma linguagem funcional (MOTIK; PATEL-SHNEIDER; PARSIA, 2012).

Na parte inferior estão as duas especificações semânticas que definem o significado das ontologias em OWL2: a semântica direta (MOTIK; PATEL-SCHNEIDER; GRAU, 2012) e a semântica baseada em RDF (SCHNEIDER, 2012). Elas são basicamente utilizadas por ferramentas para realização de consultas e recuperação de instâncias. Ambas semânticas são relacionadas através do teorema de correspondência definido no documento de semântica baseada em RDF (SCHNEIDER, 2012).

A título de exemplificação, apresentamos na Listagem 2.1 um trecho do código em OWL2 da RDBS-OWL, utilizando a sintaxe RDF/XML. Os conceitos ali definidos são **Column** e **Check Constraint**, bem como a relação *specifies* que existe entre eles. É possível ver sua representação gráfica na Figura 8. No código há também uma instância do conceito **Column** representando uma coluna identificada por `timestamp`.

Listagem 2.1 – Trecho do código OWL2 da RDBS-OWL

```
1 <owl:Class rdf:about="rdbs-o.owl#RDBS-O::Column"/>
2
3 <owl:Class rdf:about="rdbs-o.owl#RDBS-O::CheckConstraint">
4   <rdfs:subClassOf rdf:resource="rdbs-o.owl#RDBS-O::ColumnConstraint"/>
5 </owl:Class>
```

```
6
7 <owl:ObjectProperty rdf:about="rdbs-o.owl#RDBS-O::specifiesColumn">
8   <rdfs:domain rdf:resource="rdbs-o.owl#RDBS-O::CheckConstraint"/>
9   <rdfs:range rdf:resource="rdbs-o.owl#RDBS-O::Column"/>
10 </owl:ObjectProperty>
11
12 <owl:NamedIndividual rdf:about="rdbs-o.owl#timestamp">
13   <rdf:type rdf:resource="rdbs-o.owl#RDBS-O::Column"/>
14 </owl:NamedIndividual>
```

Já na Listagem 2.2 é apresentado um trecho de código OWL2 que representa os conceitos **Subclass**, **Superclass**, **Inheritance** de OOC-OWL, e as relações *inheritsIn* e *inheritedIn*. A representação gráfica em OntoUML desses conceitos estão na Figura 11.

Listagem 2.2 – Trecho do código OWL2 da OOC-OWL

```
1
2 <owl:Class rdf:about="ooc-o.owl#OOC-O::Subclass">
3   <rdfs:subClassOf rdf:resource="ooc-o.owl#OOC-O::Class"/>
4 </owl:Class>
5
6 <owl:Class rdf:about="ooc-o.owl#OOC-O::Superclass">
7   <rdfs:subClassOf rdf:resource="ooc-o.owl#OOC-O::Extendable_Class"/>
8 </owl:Class>
9
10 <owl:Class rdf:about="ooc-o.owl#OOC-O::Inheritance"/>
11
12 <owl:ObjectProperty rdf:about="ooc-o.owl#OOC-O::inheritedIn">
13   <rdfs:domain rdf:resource="ooc-o.owl#OOC-O::Superclass"/>
14   <rdfs:range rdf:resource="ooc-o.owl#OOC-O::Inheritance"/>
15 </owl:ObjectProperty>
16
17 <owl:ObjectProperty rdf:about="ooc-o.owl#OOC-O::inheritsIn">
18   <rdfs:domain rdf:resource="ooc-o.owl#OOC-O::Subclass"/>
19   <rdfs:range rdf:resource="ooc-o.owl#OOC-O::Inheritance"/>
20 </owl:ObjectProperty>
```

2.5 Trabalhos Relacionados

Diferentes propostas utilizam ontologias nos domínio de orientação a objetos ou dados relacionais como ferramentas para solução de problemas de pesquisa. Existem trabalhos como ontologias elaboradas para publicação automática de dados semânticos a partir de banco de dados (TRINH; BARKER; ALHAJJ, 2006), geração de banco de dados a partir de ontologia (LABORDA; CONRAD, 2005), modelagem de software (EVERMANN; WAND, 2005) e metodologia de desenvolvimento de software (PASTOR, 1992) apoiados em ontologia de objetos. Porém poucas pesquisas se aprofundaram na relação existente entre os domínios dos paradigmas de orientação-objeto e relacional.

Calero et al. (2006), com a finalidade de formalizar os elementos do padrão SQL:2003 e identificar suas inconsistências, apresentam uma ontologia para o recurso objeto-relacional

de tal padrão. Os autores representam a ontologia em UML e regras OCL e a subdivide em *DataTypes* (conceitos pertinentes a tipo de dados) e *SchemaObjects* (conceitos pertinentes ao contexto geral, tabelas, restrições e colunas). Ainda assim, embora o propósito dessa ontologia seja representar aspectos objeto-relacionais de uma esquema de banco de dados, a ontologia não explora essa relação entre os dois domínios e apenas representa os conceitos relacionais na forma de um diagrama de classes.

Um metamodelo sobre *data warehouse* é apresentado em (CHANG; IYENGAR et al., 2001). Ali, há uma divisão entre *Object Model* (conceitos base relacionados a classe e objeto) e *Resource* (conceitos relacionados a representação relacional). O modelo é representado em UML e define relações de especialização entre os domínios de objeto e dados relacionais. Sendo assim, *Column* e *Data Type* do submodelo *Relational* são especializações de *Attribute* e *Classifier* do submodelo *Object Model*, respectivamente. Não há, porém, uma base em ontologia de fundamentação e relação por mapeamento em linguagem de programação.

Uma linguagem interoperável é proposta por Schaub e Malloy (2016). A linguagem proposta pelos autores, denominada iJava, foi mapeada para Java, Python, e C++, permitindo a migração de código entre estas linguagens de programação. O uso de uma linguagem intermediária com o papel de *interlingua* facilita a inclusão de novas linguagens, uma vez que basta efetuar o mapeamento entre a linguagem e a *interlingua*, descartando a necessidade de mapear a nova linguagem a todas as outras já incluídas. Porém, iJava não possui um modelo conceitual formal e explícito e não inclui na conversão entre as linguagens os mapeamentos objeto/relacionais.

3 Proposta do Trabalho

Esse capítulo apresenta a Ontologia de Mapeamento Objeto/Relacional, proposta nessa dissertação. A ORM-O (*Object/Relational Mapping Ontology*) é uma ontologia de referência no domínio de *frameworks* ORM e visa identificar e representar os conceitos do mapeamento objeto/relacional.

A ORM-O foi construída no escopo de um projeto que visa criar uma rede de ontologias sobre *frameworks* de desenvolvimento de software.[1] Tais ontologias nos permitirão automatizar tarefas de interoperabilidade semântica, como migração de código entre *frameworks*, ou ainda especificar *smells* na arquitetura do software de forma independente de *framework* ou linguagem, por exemplo.

É importante ressaltar ainda que os conceitos representados pela ORM-O dizem respeito ao escopo do código-fonte, o que implica que os padrões de configuração dos *frameworks* como banco de dados e com o sistema gerenciador de banco de dados não são por ela cobertos.

A construção da ontologia foi baseada no método SABiO (FALBO, 2014), que define as fases de desenvolvimento descritas na Seção 2.2.2. A fase de identificação do propósito e elicitação de requisitos nos levou a uma série de questões de competência e requisitos não-funcionais. As questões de competência identificadas estão listadas a seguir:

- QC01: Que classes são mapeadas para o banco de dados?
- QC02: Como os relacionamentos entre classes são mapeados para o banco de dados?
- QC03: Que atributos de uma dada classe são mapeados para o banco de dados?
- QC04: Que atributos de uma dada classe são mapeados para chave primária no banco de dados?
- QC05: Que atributos de uma dada classe são mapeados para chave estrangeira no banco de dados?
- QC06: Como os relacionamentos de herança entre classes são mapeados para o banco de dados?

Os requisitos não-funcionais tiveram como embasamento o reúso e a cobertura da ontologia, e estão apresentados abaixo:

[1] *Software Frameworks Ontology Network*: <https://nemo.inf.ufes.br/projects/sfwon/>

- RNF01: A ontologia deve considerar *frameworks* de diversas linguagens de programação e diferentes *frameworks* para uma mesma linguagem de programação;
- RNF02: A ontologia deve ser baseada em uma ontologia de fundamentação;
- RNF03: A ontologia deve reutilizar ontologias existentes de assuntos relacionados;
- RNF04: A ontologia deverá ser modularizada para permitir seu reuso.

A fase de captura e formalização da ontologia que representou o modelo gráfico da conceituação do domínio foi feita em OntoUML (GUIZZARDI, 2005), descrita na Seção 2.2.1. Essa fase foi apoiada por um processo de aquisição de conhecimento que considerou diferentes linguagens e *frameworks*. As linguagens escolhidas foram Java, Python e C++. Essa escolha foi realizada com base nos ranqueamentos TIOBE,[2] IEE Spectrum[3] e Redmonk,[4] já utilizados por Aguiar, Falbo e Souza (2019) para a construção da OOC-O (cf. Seção 2.3.2). Já os *frameworks*, foram escolhidos com base na popularidade do *Stack Overflow*,[5] um site de perguntas e respostas amplamente conhecido e utilizado pela comunidade de desenvolvedores.

Sendo assim, obedecendo RNF01 e RNF02, foram selecionados os seguintes *frameworks*: Java Persistence API (JPA) (JPA, 2019), que é o padrão de implementação de *frameworks* para Java, como o Hibernate; Django (DJANGO, 2019) e SQLAlchemy (SQLALCHEMY, 2019), que são *frameworks* para a linguagem Python; e QxORM (QXORM, 2019) e ODB (ODB, 2019) que são *frameworks* para C++.

Atendendo RNF03, foram reutilizadas as ontologias RDBS-O e OOC-O, apresentadas na Seção 2.3. Por fim, para melhor desenvolvimento, representação e entendimento, e em conformidade à RNF04, a ORM-O foi dividida em quatro partes nomeadas ORM-O Class, ORM-O Variable, ORM-O Relationship e ORM-O Inheritance. A Figura 14 apresenta a composição da ORM-O e também sua relação com as ontologias RBDS-O e OOC-O.

O restante desse capítulo é dedicado à apresentação das quatro subontologias de ORM-O. A Seção 3.1 descreve a **ORM-O Class**, que trata do mapeamento individual das classes; a Seção 3.2 introduz a **ORM-O Variable**, que trata do mapeamento de variáveis (atributos); a Seção 3.3 apresenta a **ORM-O Relationship**, que trata do mapeamento das relações entre classes; por fim, a Seção 3.4 apresenta a **ORM-O Inheritance**, que mostra os conceitos do mapeamento de heranças entre classes.

[2] <http://tiobe.com>
[3] <http://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages>
[4] <http://redmonk.com/sogrady/2019/03/20/language-rankings-1-19/>
[5] <https://stackoverflow.com/tags>

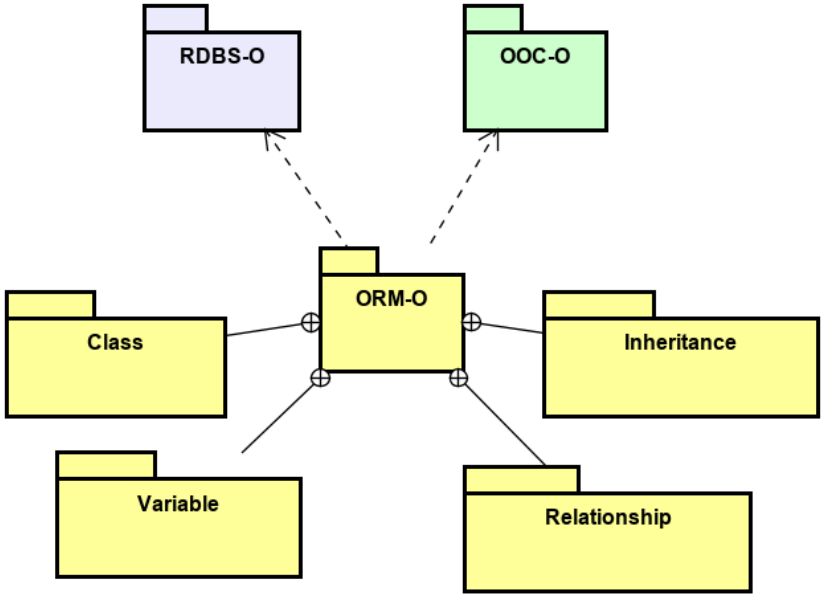

Figura 14 – Composição da ORM-O e sua associação com OOC-O e RDBS-O.

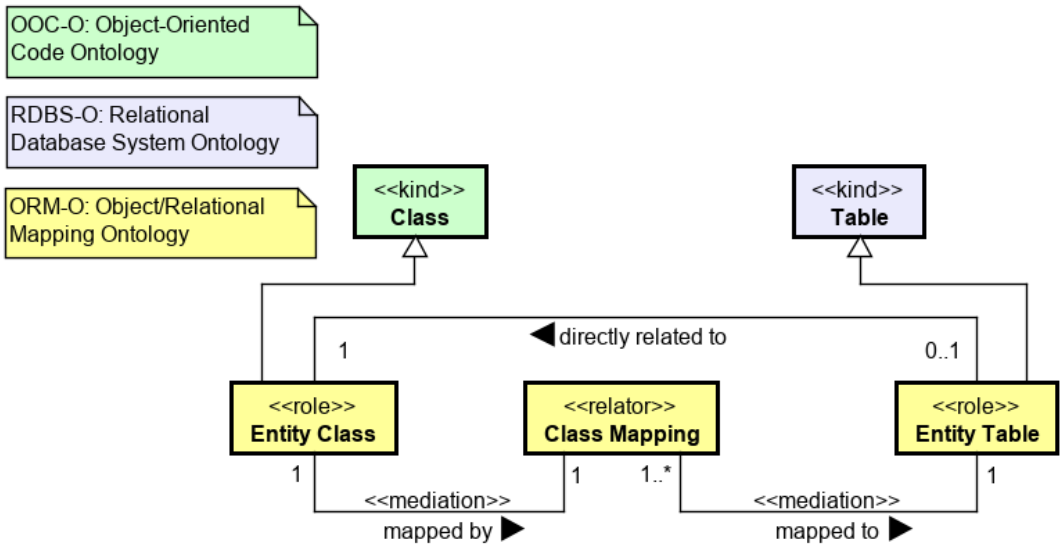

Figura 15 – Subontologia ORM-O Class.

3.1 ORM-O Class

Nessa seção apresentamos a subontologia ORM-O Class. Essa subontologia utiliza apenas dois conceitos da OOC-O e RDBS-O: **Class** e **Table**, respectivamente. O modelo em OntoUML desenvolvido pode ser visto na Figura 15.

Para representar classes que desempenham o papel de classes mapeadas, **Class** foi especializado em **Entity Class**. De maneira análoga, **Table** foi especializada em **Entity Table**, conceituando assim tabelas do banco de dados que persistem dados de instâncias de classes (objetos).

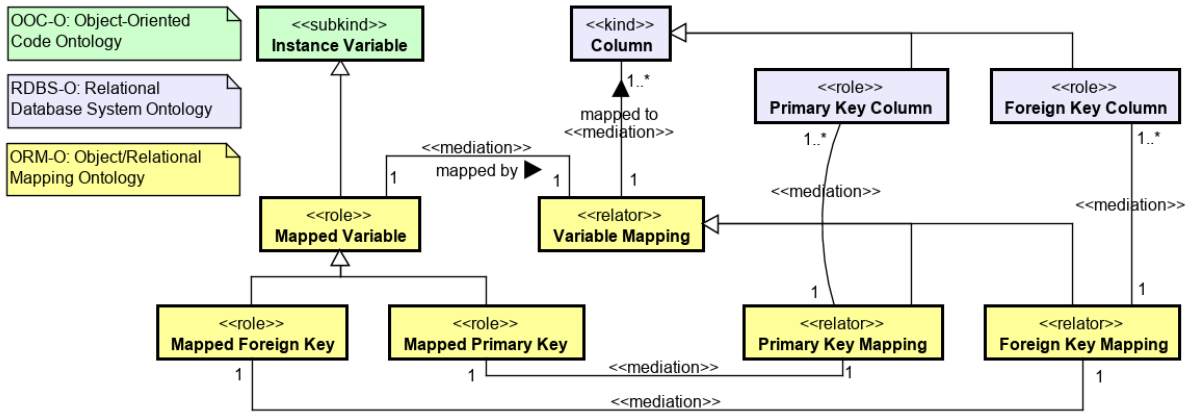

Figura 16 – Subontologia ORM-O Variable.

O `<<relator>>` **Class Mapping** representa o conceito do mapeamento de OO para relacional. Uma vez que uma classe assume o papel de **Entity Class** ela será mapeada, por isso a cardinalidade `1` na relação *mapped by*. A cardinalidade oposta dessa relação também é `1`, implicando assim que um **Class Mapping** é exclusivo para a **Entity Class**.

Já as cardinalidades da relação *mapped to* implicam que um **Class Mapping** mapeia a classe para uma única **Entity Table** porém uma **Entity Table** pode persistir informações de mais de uma única classe, por isso a cardinalidade `1..*` do lado do **Class Mapping**. O motivo para isso são os mapeamentos de herança, detalhados na Seção 3.4.

Por fim, uma **Entity Table** tem uma relação direta com uma **Entity Class**, representada no modelo por *directly related to*. Essa relação representa o conceito de um classe que originou a tabela para onde ela será mapeada. O fato de nem toda classe originar uma tabela no banco de dados explica a cardinalidade `0..1` do lado de **Entity Table**, porém toda tabela é originada de uma classe, justificando assim a cardinalidade `1` no lado de **Entity Class**.

3.2 ORM-O Variable

Essa seção apresenta a subontologia ORM-O Variable. A Figura 16 mostra o modelo OntoUML que representa os conceitos dessa subontologia.

Os atributos de uma classe que podem ser mapeados são representados pelo conceito **Instance Variable**, oriundo da OOC-O. Sua especialização **Mapped Variable** é o conceito das variáveis que são efetivamente mapeadas do código OO para o banco de dados relacional. O conceito do mapeamento propriamente dito é representado pelo `<<relator>>` **Variable Mapping**. Uma vez que uma variável assume o papel de **Mapped Variable**, ela possui uma relação `<<mediation>>` **mappedy by** com um **Variable Mapping** e essa restrição é percebida pela cardinalidade `1`.

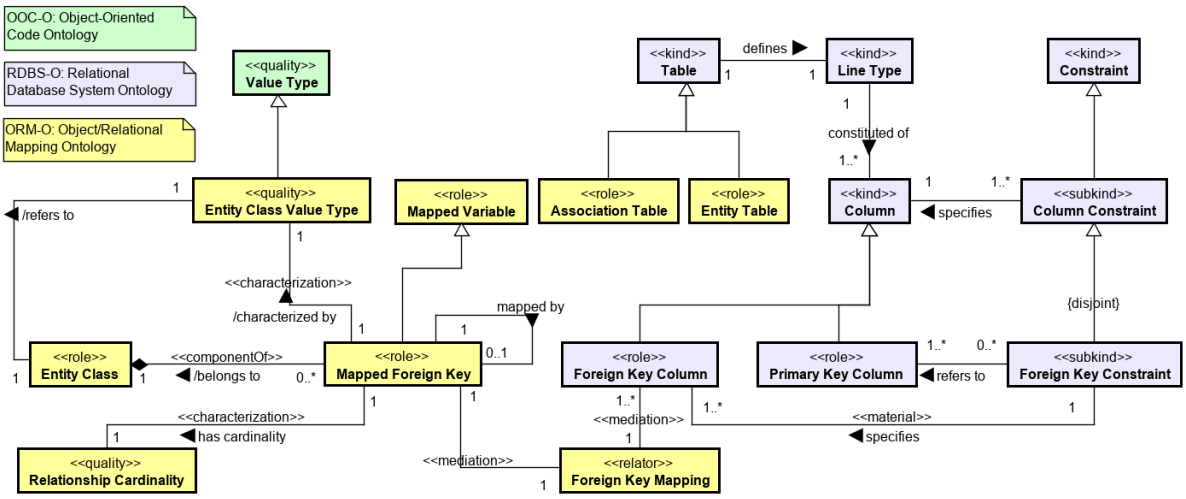

Figura 17 – Subontologia ORM-O Relationship.

Atributos são mapeados para colunas no banco de dados. Por esse fato, o `<<relator>>` **Variable Mapping** tem relação *mapped to* com **Column**, que é conceito da RDBS-O. A cardinalidade `1..*` observada na relação *mapped to* entre **Variable Mapping** e **Column** advém da possibilidade de uma variável ser persistida em uma ou mais colunas da tabela do banco de dados.

Alguns atributos de classes de um código OO são mapeados para colunas com papéis específicos. Essas colunas são representadas pelos conceitos **Primary Key Column** e **Foreign Key Column**. Os atributos que são para elas mapeados são representados pelos conceitos **Mapped Primary Key** e **Mapped Foreign Key**, que especializam **Mapped Variable**.

Seguindo a linha das especificidades de chave primária e chave estrangeira, o `<<relator>>` **Variable Mapping** também é especializado em **Primary Key Mapping** e **Foreign Key Mapping**, derivando implicitamente as relações *mapped by* e *mapped to*.

O conceito **Mapped Foreign Key** é extremamente importante para que seja possível conceituar o mapeamento de relacionamento entre classes do código OO para o relacionamento de tabelas no banco de dados. Essa conceituação é representada na subontologia ORM-O Relationship, a seguir.

3.3 ORM-O Relationship

Essa seção apresenta a subontologia ORM-O Relationship, que é a conceituação dos mapeamento de relacionamentos do código OO para tabelas do banco de dados relacional. A Figura 17 mostra o modelo OntoUML que representa esses conceitos.

Um peça chave para o entendimento e representação é o **Mapped Foreign Key**, que é um conceito já explicado na Seção 3.2. Uma **Mapped Foreign Key** e mapeada para

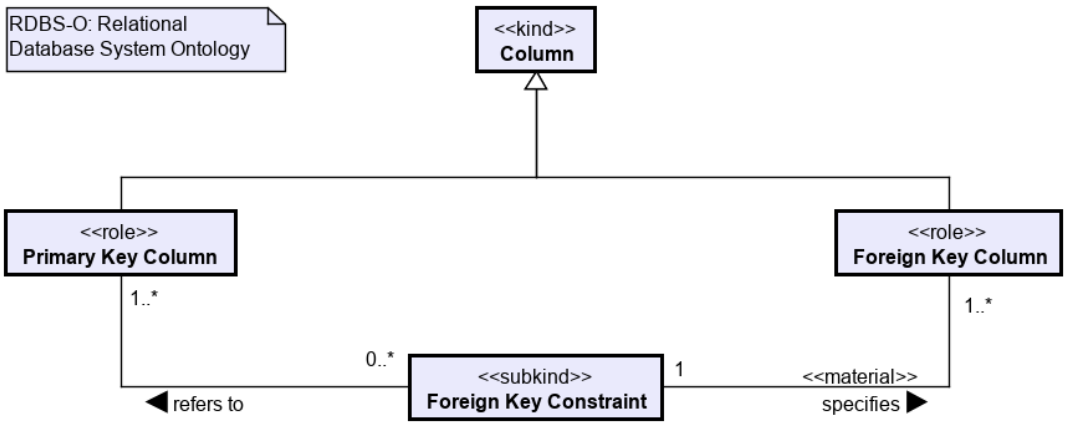

Figura 18 – Relação entre Primary Key Column e Foreign Key Column.

uma **Foreign Key Column** e, na RDBS-O, há relações entre conceitos que associam uma **Foreign Key Column** a uma **Primary Key Column**. Olhando o recorte da RDBS-O apresentado na Figura 18, podemos notar que uma **Foreign Key Constraint** tem uma relação `<<material>>` *specifies* com **Foreign Key Column** ao mesmo tempo em que tem uma relação *refers to* com uma **Primary Key Column**.

No lado do código OO, o relacionamento é notado através de relações derivadas pelas especializações de conceitos. Essas relações derivadas estão explícitas na Figura 19. Uma vez que a chave estrangeira se refere a uma classe que que também é mapeada para o banco de dados, surge o **Entity Class Value Type** que é uma especialização de **Value Type** cuja relação *refers to* é feita obrigatoriamente com uma **Entity Class**. A **Mapped Foreign Key**, por sua vez, deriva as relações `<<characterization>>` *characterized by* com **Entity Class Value Type**, que é derivada da relação entre **Variable** e **Value Type**. A última derivação é a `<<componentOf>>` *belongs to* entre **Mapped Foreign Key** e **Entity Class** que advém da relação entre **Member** e **Class**.

Como pode ser observado na Figura 17 há ainda o conceito **Relationship Cardinality**. É sabido que, no paradigma relacional, existem quatro diferentes cardinalidades de relacionamentos: um para um (*one-to-one*), um para muitos (*one-to-many*), muitos para um (*many-to-one*) e muitos para muitos (*many-to-many*). Para que o mapeamento objeto/relacional seja feito corretamente, há a identificação em código para que o *framework* saiba como mapear corretamente. sendo assim, **Mapped Foreign Key** tem uma relação `<<characterization>>` *has cardinality* com **Relationship Cardinality**.

Percebemos ainda que, no lado OO, dizer que uma classe A tem um relacionamento um para um com uma classe B não implica necessariamente em dizer que a classe B tem o mesmo relacionamento com A. No código OO deve ser definido um segundo relacionamento de B para A e, para que o *framework* entenda que se trata do mesmo relacionamento e faça o mapeamento corretamente, essa bidirecionalidade deve ser explicitada. Esse conceito é

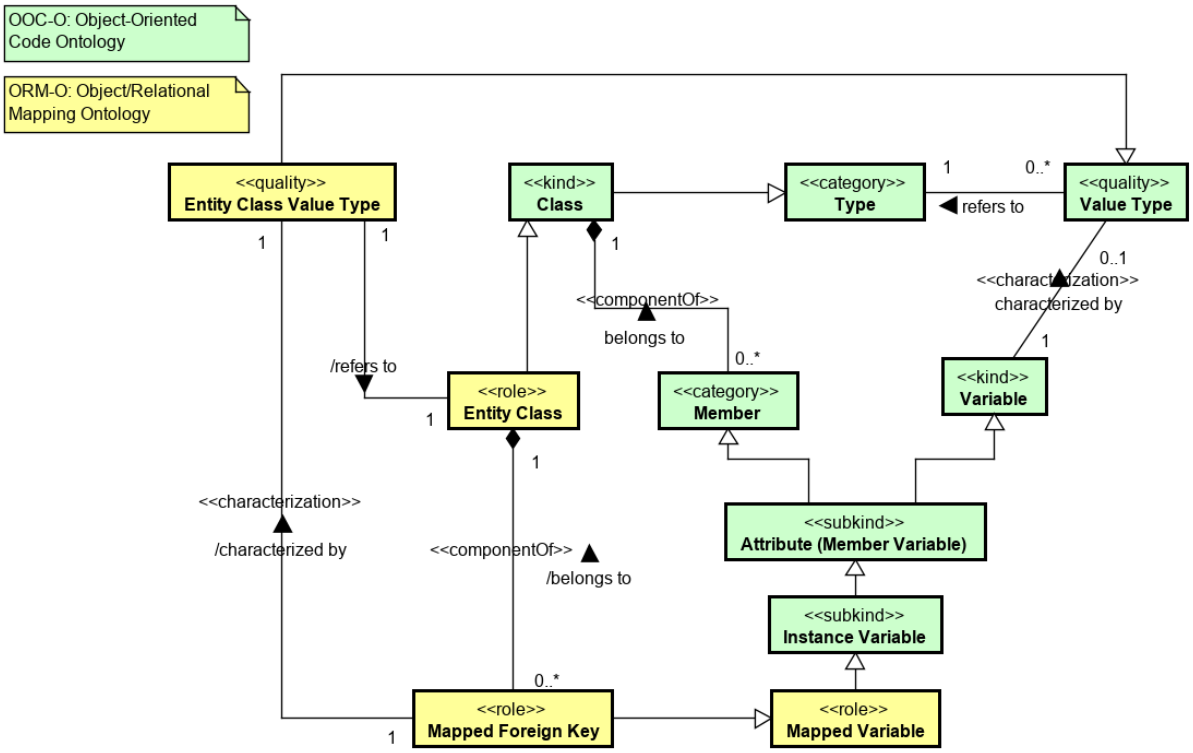

Figura 19 – Relações derivadas por Mapped Foreign Key.

representado pela relação *inverse of*, que pode associar uma **Mapped Foreign Key** a outra.

3.4 ORM-O Inheritance

Nessa seção apresentaremos a subontologia ORM-O Inheritance, que é a conceituação do mapeamento de herança entre os paradigmas OO e relacional.

Para um melhor entendimento da representação é preciso reforçar que existem três diferentes estratégias de mapeamento de herança (AMBLER, 2010), a saber:

- A estratégia *Single Table* é aquela em que os objetos das classes envolvidas em uma hierarquia são persistidos em uma única tabela no banco de dados;
- Já na estratégia *Table per Class* cada classe da hierarquia possui sua própria tabela e ali são armazenadas as informações dos atributos dessa única classe. Isso implica dizer que para recuperar informações de uma subclasse é necessário unir informações de diferentes tabelas do banco de dados;
- Por último, na estratégia *Table per Concrete Class* há uma tabela no banco de dados para cada classe concreta da hierarquia (isto é, classes que podem ser instanciadas) e tal tabela deve conter toda informação necessária para recuperar a informação de

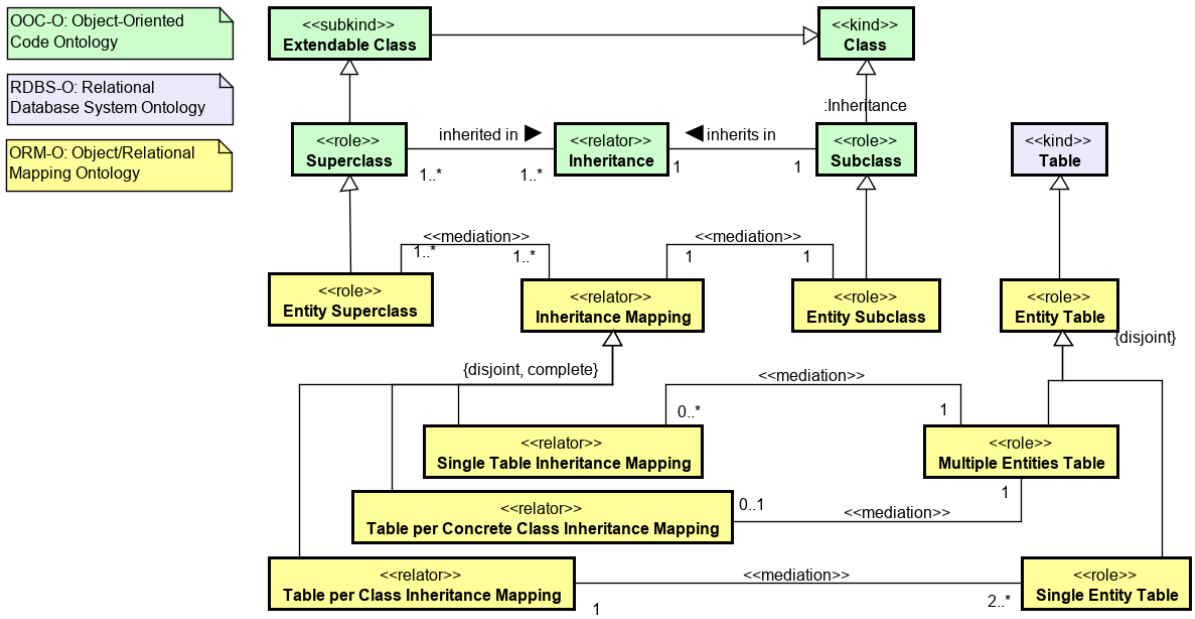

Figura 20 – Modelo OntoUML da subontologia ORM-O Inheritance.

um objeto, implicando na inclusão de colunas referentes a atributos das superclasses.

Uma vez entendidas as possibilidades de estratégia de mapeamento de herança, apresentamos na Figura 20 o modelo conceitual em OntoUML da ORM-O Inheritance. Nessa subontologia são reutilizados os conceitos **Class**, **Extendable Class**, **Superclass**, **Subclass** e **Inheritance** da OOC-O, o conceito **Table** da RDBS-O e também o conceito de **Entity Table** da própria ORM-O que já foi discutido na Seção 3.1.

Assim como a especialização de **Class** em **Entity Class** apresentada na subontologia descrita na Seção 3.1, aqui temos as especializações de **Superclass** e **Subclass** em **Entity Superclass** e **Entity Subclass** respectivamente. Tais especializações são necessárias para representar superclasses ou subclasses que são mapeadas para o banco de dados.

Para a representação do mapeamento de herança o conceito **Inheritance Mapping** é introduzido e associa os conceitos recém descritos. Uma vez que uma superclasse pode estar envolvida em diferentes hierarquias e uma hierarquia tem, necessariamente, uma ou mais superclasses a cardinalidade entre **Entity Superclass** e **Inheritance Mapping** é `1..*` nos dois lados.

Já a cardinalidade entre **Inheritance Mapping** e **Entity Subclass** é de `1` dos dois lados pelo fato de que, no papel de subclasse, uma classe participa de uma única hierarquia.

Respeitando as diferentes estratégias de mapeamento de herança, **Inheritance Mapping** é especializado em **Single Table Inheritance Mapping**, **Table Per Class**

Inheritance Mapping e **Table Per Concrete Class Inheritance Mapping**, que formam um *generalization set* completo e disjunto.

Além disso, a diferença entre as estratégias de mapeamento quanto à tabela em que o objeto será persistido gerou a necessidade de especialiação de **Entity Table** em **Single Entity Table** e **Multiple Entities Table**. A primeira trata de quando uma tabela assume o papel de armazenar informações dos objetos de uma única classe. De maneira oposta, na segunda são persistidos os atributos de duas ou mais classes da hierarquia.

Sendo assim, como a estratégia *Single Table* implica que todas as classes da hierarquias são mapeadas para uma única tabela, **Single Table Inheritance Mapping** é associado com **Multiple Entities Table** com cardinalidade `1`. Seguindo a análise, **Table per Class Inheritance Mapping** mapeia cada classe da hierarquia para uma tabela específica para ela, portanto é associado com **Single Entity Table** com cardinalidade `2..*`. Por fim, a estratégia *Table per Concrete Class*, que implica que toda a informação de um objeto esteja em uma única tabela, mapeia as instâncias de uma **Entity Subclass** para uma única **Multiple Entities Table**, pois contém também os atributos das **Entity Superclasses**.

4 Avaliação do Trabalho

A avaliação da ORM-O foi realizada por meio de atividades de verificação e validação, seguindo o método SABiO (FALBO, 2014). Além disso, como prova de conceito, a ontologia foi utilizada na prática como interlíngua em uma ferramenta de migração de código que foi desenvolvida a partir dela. Este capítulo descreve e apresenta essas etapas da avaliação.

Na Seção 4.1 está a atividade de verificação, que identifica a resposta das questões de competência. Na Seção 4.2, que apresenta a atividade de validação, mostramos os conceitos da ontologia instanciados com um dos *frameworks* analisados na atividade de aquisição de conhecimento. Ao fim do capítulo, na Seção 4.3 descrevemos a ferramenta de migração que é baseada na ORM-O.

4.1 Verificação

A atividade de verificação foi realizada através da identificação dos conceitos que compõem a ontologia e respondem às questões de competências identificadas e apresentadas no Capítulo 3. A Tabela 2 apresenta as respostas das QCs levantadas no capítulo supracitado, mostrando quais conceitos e relações são utilizados para responder cada questão.

Tabela 2 – Respostas às Questões de Competência.

ID	Resposta
QC01	Que classes são mapeadas para o banco de dados? **Entity Class** *subtype of* **Class** **Class Mapping** *mapped by* **Entity Class** **Class Mapping** *mapped to* **Entity Table** **Entity Table** *subtype of* **Table**
	continua na próxima página

Tabela 2. Continuação da página anterior

ID	Resposta
QC02	Como os relacionamentos entre classes são mapeados para o banco de dados? **Mapped Foreing Key** */belongs to* **Entity Class** **Mapped Foreign Key** */characterized by* **Entity Class Value Type** **Entity Class Value Type** */refers to* **Entity Class** **Mapped Foreign Ke**y *mapped by* **Foreign Key Mapping** **Foreing Key Mapping** *mapped to* **Foreign Key Column** **Foreign Key Contraint** *specifies* **Foreign Key Column** **Foreign Key Constraint** *refers to* **Primary Key Column**
QC03	Que atributos de uma dada classe são mapeados para o banco de dados? **Mapped Variable** *subtype of* **Instance Variable** **Mapped Variable** *mediation* **Variable Mapping** **Variable Mapping** *mediation* **Column**
QC04	Que atributos de uma dada classe são mapeados para chave primária no banco de dados? **Mapped Variable** *subtype of* **Instance Variable** **Mapped Primary Key** *subtype of* **Mapped Variable** **Mapped Primary Key** *mediation* **Primary Key Mapping** P**rimary Key Mapping** *mediation* **Primary Key Column**
QC05	Que atributos de uma dada classe são mapeados para chave estrangeira no banco de dados? **Mapped Variable** *subtype of* **Instance Variable** **Mapped Foreign Key** *subtype of* **Mapped Variable** **Mapped Foreign Key** *mediation* **Foreign Key Mapping** **Foreign Key Mapping** *mediation* **Foreign Key Column**
QC06	Como os relacionamentos de herança entre classes são mapeados para o banco de dados? **Entity Subclass** *subtype of* **Subclass** **Entity Subclass** *mediation* **Inheritance Mapping** **Inheritance Mapping** *mediation* **Entity Superclass** **Entity Superclass** *subtype of* **Superclass** **Single Table Inheritance Mapping**, **Table per Class Inheritance Mapping** and **Table per Subclass Inheritance Mapping** *subtype of* **Inheritance Mapping**; *mediation* **Entity Table** **Entity Table** *subtype of* **Table**

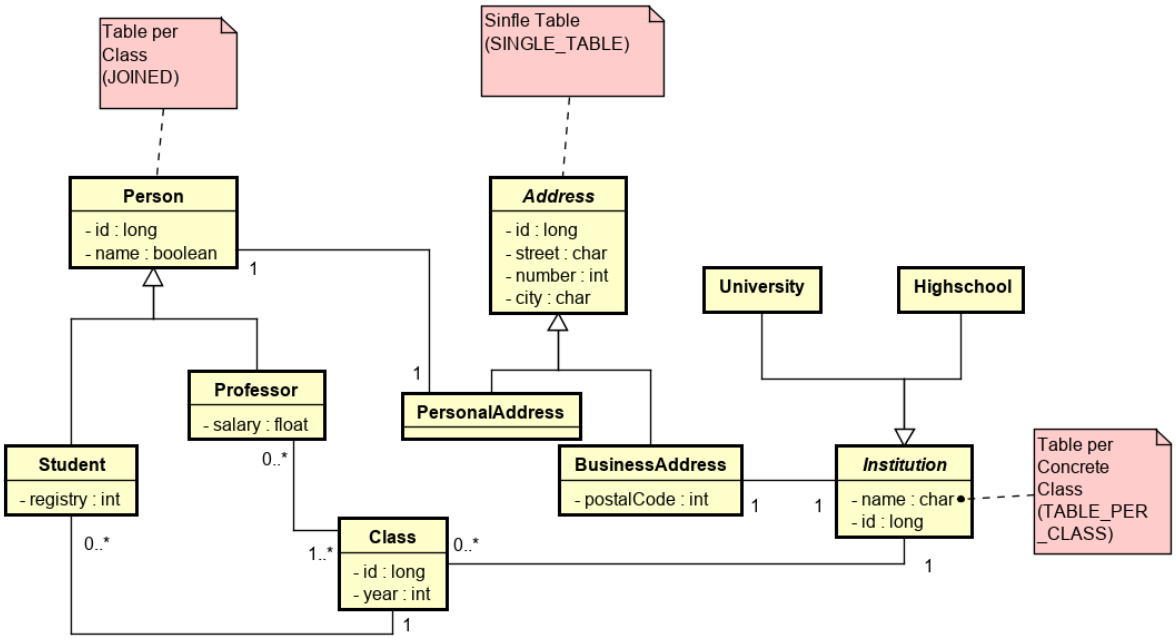

Figura 21 – Modelo de classes do exemplo de domínio a ser instanciado pela ontologia.

4.2 Validação

A validação da ORM-O foi realizada por meio da instanciação dos conceitos da ontologia utilizando o *framework Hibernate*, da linguagem Java. Tal *framework* é uma implementação do JPA, cuja documentação foi incluída nos estudos para aquisição de conhecimento prévio para construção da ontologia.

Foi criado um domínio fictício para que a codificação equivalente à instanciação da ontologia fosse possível. O modelo de classes de tal domínio está representado na Figura 21. O banco de dados onde os objetos das classes apresentadas serão persistidos está representado pelo modelo de entidades-relacionamento apresentado na Figura 22. Vale lembrar que, por padrão, o *Hibernate* cria e gerencia as tabelas com os nomes em minúsculo (lowercase), por isso a diferença notada entre os nomes das classes da Figura 21 e os nomes das tabelas na Figura 22.

A seguir serão apresentadas listagens de trechos do código do domínio e a identificação dos conceitos da ontologia que são por ele instanciados. A implementação completa do código está disponível em <https://github.com/nemo-ufes/ORMMigrationTool/tree/master/ExampleCode/Java>.

O trecho de código da classe `Person` apresentado na Listagem 4.1 demonstra uma instância do conceito **Entity Class** uma vez que, em Java, o uso da anotação `@Entity` do *Hibernate* implica que a classe será persistida no banco de dados, portanto será mapeada.

Além disso, a existência da **Entity Class** gera implicitamente uma instância do

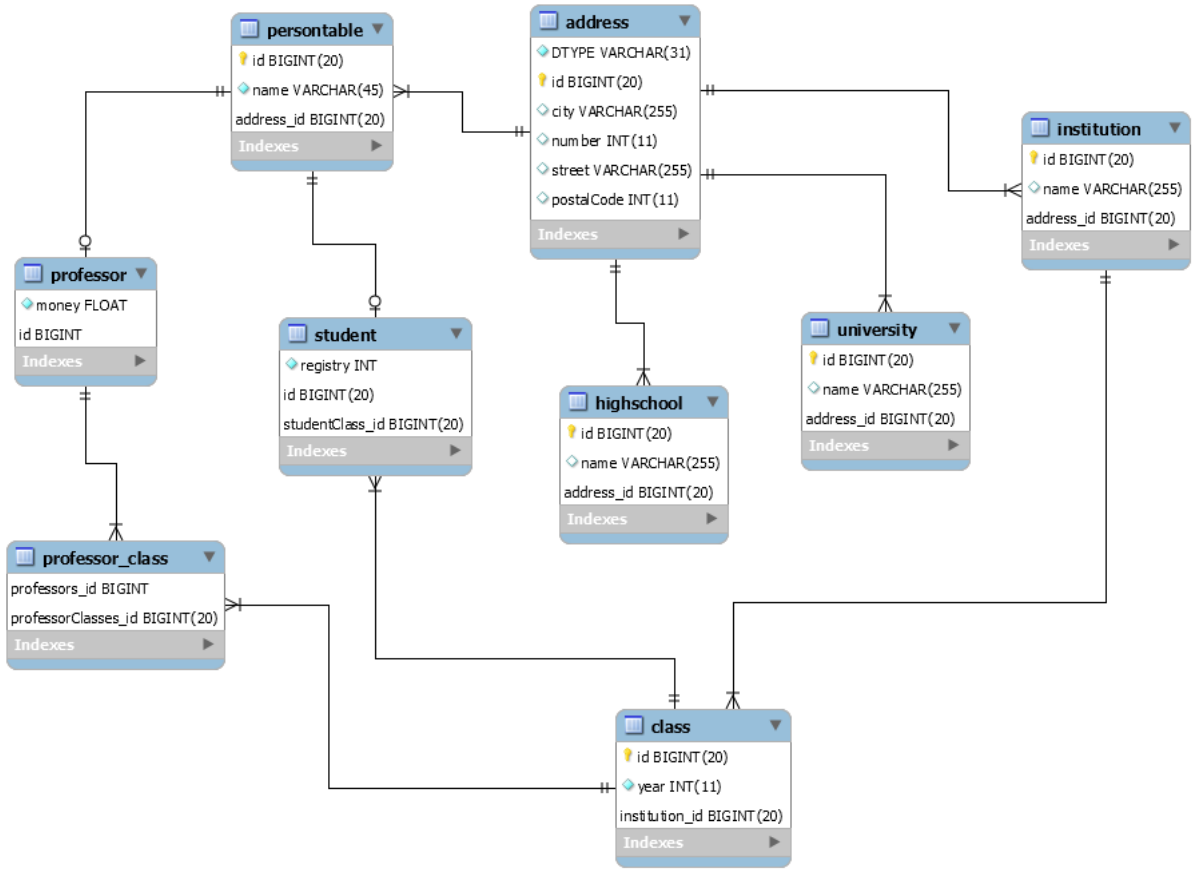

Figura 22 – Modelo ER do banco de dados onde os objetos serão persistidos

Listagem 4.1 – Trecho do código com instância de Entity Class

```
1 @Entity
2 public class Person{...};
```

conceito **Class Mapping** que relaciona a classe a ser persistida com uma tabela do banco de dados, que por sua vez assume o papel de **Entity Table**. Vale ressaltar ainda que no caso do *framework* escolhido a definição de qual tabela do banco de dados assumirá esse papel é feita de maneira automática através do nome da **Entity Class**, porém isso pode ser definido de maneira explícita como demonstrado na Listagem 4.2, onde a anotação `@Table` é utilizada.

Listagem 4.2 – Trecho do código com instância de Entity Table

```
1 @Entity
2 @Table(name="persontable")
3 public class Person{...};
```

Já uma instância do conceito **Mapped Variable** pode ser observada na Listagem 4.3. Em *Hibernate* qualquer atributo de uma **Entity Class** que não seja precedido da anotação `@Transient` é uma instância de **Mapped Variable**.

Listagem 4.3 – Trecho do código com instância de Mapped Variable

```
1 @Entity
2 public class Professor extends Person {
3   ...
4   @Column(name="money")
5   private float salary;
6   ...
7 }
```

Similarmente como ocorre com **Entity Class**, a existência de uma **Mapped Variable** gera uma instância do conceito **Variable Mapping**, que é o que relaciona o atributo com **Column** que será mapeado. Automaticamente a coluna será a de mesmo nome do atributo, a não ser que haja uma indicação dizendo algo diferente, que é o que ocorre com o atributo precedido da anotação `@Column` na Listagem 4.3.

As instâncias de especializações de **Mapped Variable** e de **Variable Mapping** podem ser exemplificadas no trecho de código da Listagem 4.4. Ali é possível notar a presença da anotação `@Id`, que indica que o atributo é uma chave primária e gera a existência de uma **Primary Key Mapping**. Já a anotação `@OneToOne` atribui o papel de chave estrangeira ao atributo e, analogamente, gera uma **Foreign Key Mapping**.

Listagem 4.4 – Trecho do código com instância de Mapped Primary Key e Mapped Foreing Key

```
1 @Entity
2 public class Person {
3   ...
4   @Id
5   private Long id;
6
7   @OneToOne
8   private PersonalAddress address;
9   ...
10 }
```

Como explicitado na Seção 3.3, o conceito **Mapped Foreign Key** é de extrema importância para a conceituação dos relacionamentos. Para tal, ele tem uma relação *characterized by* com um **Entity Class Value Type**, que na Listagem 4.4 é exemplificado com a declaração `private PersonalAddress address`, implicando que a **Mapped Foreign Key** `address` é caracterizada por uma instância de **Entity Class Value Type** que se refere a **Entity Class** `PersonalAddress`.

Além disso, **Relationship Cardinality** é instanciado com uma das quatro possibilidades de cardinalidade e, ainda na Listagem 4.4, podemos ver a anotação `@OneToOne`, indicando que a cardinalidade, nesse caso, é de um para um.

Ainda sobre relacionamentos, podemos exemplificar instâncias de **Mapped Foreign Key** que possuam a relação *inverse of*. Tais exemplos de instâncias estão demons-

tradas no trecho de código mostrado na Listagem 4.5.

Listagem 4.5 – Trecho do código com instâncias de Mapped Foreing Key com a relação inverse of

```
1  @Entity
2  public class Class {
3    ...
4    @ManyToMany(mappedBy="professorClasses")
5    private List<Professor> professors;
6    ...
7  }
8
9  @Entity
10 public class Professor extends Person {
11   ...
12   @ManyToMany
13   private List<Class> professorClasses;
14   ...
15 }
```

Na listagem é possível observar as instâncias de **Mapped Foreing Key** pelas anotações `@ManyToMany`. Adicionalmente é possível observar em uma das anotações um acréscimo do atributo `mappedBy`. Dessa maneira o *Hibernate* identifica que os relacionamentos no código OO entre as classes `Class` e `Professor` indicados pelas respectivas chaves estrangeiras são, no banco de dados, um único relacionamento que é bidirecional.

Uma última etapa da validação é checar instâncias dos conceitos relacionados ao mapeamento de herança. Com já dito na Seção 3.4, há três possíveis estratégias de mapeamento de herança. A Listagem 4.6 apresenta trechos de código que contém instâncias dos conceitos **Entity Superclass**, **Entity Subclass** e das três diferentes especializações de **Inheritance Mapping**.

Listagem 4.6 – Trecho do código com instâncias dos conceitos da ORM-O Inheritance

```
1
2  @Entity
3  @Inheritance(strategy = InheritanceType.SINGLE_TABLE)
4  public abstract class Address {...}
5
6  @Entity
7  public class BusinessAddress extends Address{...}
8
9  @Entity
10 @Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
11 public abstract class Institution {...}
12
13 @Entity
14 public class Highschool extends Institution{...}
15
16 @Entity
17 @Inheritance(strategy = InheritanceType.JOINED)
18 public class Person {...}
```

```
19
20 @Entity
21 public class Professor extends Person{...}
```

Nos trechos de código é possível observar que as classes `BusinessAddress`, `Highschool` e `Professor` são subclasses que serão efetivamente mapeadas, portanto são instâncias do conceito **Entity Subclass**. Analogamente as classes que elas herdam (`Address`, `Address` e `Person`) são instâncias de **Entity Superclass**.

É possível notar também a presença da anotação `@Inheritance` antecedendo a declaração das superclasses. Essa anotação indica, através do atributo `strategy` qual a estratégia de mapeamento de herança a ser adotada. Sendo assim, a hierarquia da superclasse `Address` adota a estratégia *Single Table* sendo instância do conceito **Single Table Inheritance Mapping**. Já a hierarquia da classe `Institution` adota a estratégia *Table per Concrete Class* instanciando o conceito **Table per Concrete Class Inheritance Mapping** e, finalmente, a hierarquia da classe `Person` tem *Teble per Class* como estratégia de mapeamento sendo instância do conceito **Table per Class Inheritance Mapping**.

4.3 Ferramenta de Migração

Como uma forma de avaliação da cobertura da ontologia e prova de conceito, foi desenvolvida uma ferramenta de migração de código. A funcionalidade da ferramenta é carregar uma implementação funcional da ORM-O, receber como entrada um código fonte de uma linguagem OO com utilização de um *framework* ORM e, como saída, gerar o código fonte de uma segunda linguagem OO com um de seus respectivos *frameworks* ORM. Esta seção descreve como a ferramenta foi implementada e apresenta os resultados de testes.

A versão inicial da ferramenta, cujo o desenvolvimento foi feito neste trabalho, migra código OO da linguagem de programação Java utilizando o *Hibernate*, que é uma implementação do JPA, para um código OO na linguagem Python que utiliza o *framework* Django. As etapas do fluxo de execução podem ser observados na Figura 23.

O primeiro passo para que fosse possível desenvolver a ferramenta foi implementar a ontologia em uma linguagem operacional. Implementou-se então em OWL2, linguagem operacional de ontologias já apresentada na Seção 2.4.

Para auxílio no desenvolvimento da ontologia em OWL2 utilizou-se a ferramenta Protégé (MUSEN et al., 2015), que é um editor de ontologias de código aberto e gratuito que fornece uma interface gráfica para definição de ontologias.

O primeiro passo no Protégé é carregar as ontologias reutilizadas: OOC-O e RDBS-O. A tela do Protégé indicando o carregamento das respectivas ontologias pode ser observada na Figura 24.

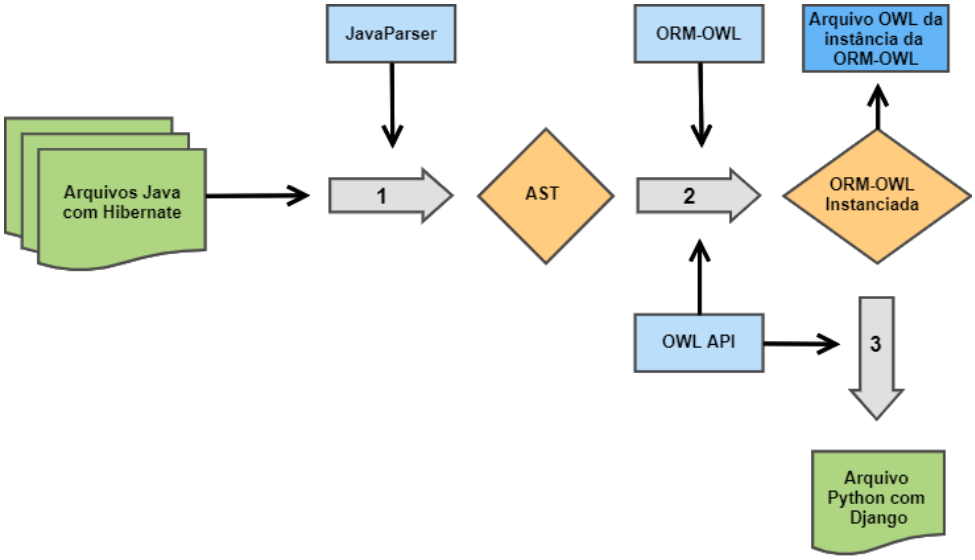

Figura 23 – Fluxo de execução da ferramenta de migração.

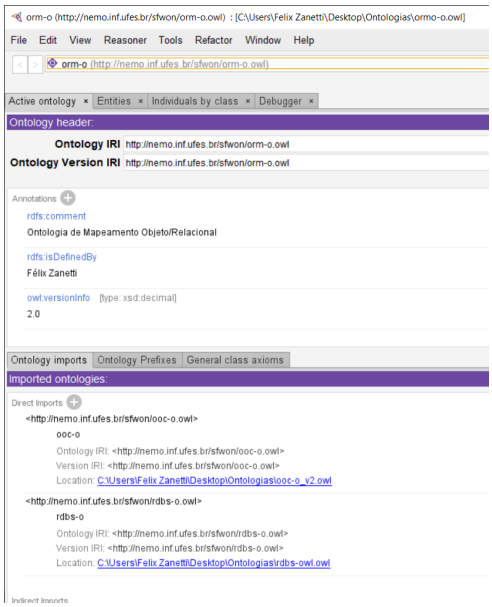

Figura 24 – Parte da tela do Protégé com o carregamento da OOC-O e RDBS-O.

Após o carregamento prévio das ontologias cujos conceitos são reutilizados, foram então incluídos os conceitos definidos pela ORM-O. A árvore de hierarquia da interface do Protégé pode ser observada na Figura 25. Tanto os arquivos da implementação operacional da OOC-O e da RDBS-O utilizados e o arquivo de extensão OWL resultante desse procedimento estão disponíveis na página do SFWON.[1]

Uma vez que a ORM-O teve sua versão operacional implementada, a ferramenta de migração pôde ter seu desenvolvimento iniciado. Optou-se por desenvolver a ferramenta na linguagem Java. O principal motivo dessa decisão é pelo fato de existirem duas ferramentas

[1] http://nemo.inf.ufes.br/projects/sfwon

Figura 25 – Árvore de hierarquia dos conceitos no Protégé.

que ajudarão nessa primeira versão: o JavaParser[2] e o OWLAPI.[3]

JavaParser é uma biblioteca para Java que processa um código em Java e monta uma árvore de sintaxe abstrata (*Abstract Syntax Tree* - AST). Com isso é possível analisar todo o código para que a ontologia possa ser instanciada. Já o OWLAPI é uma API Java de código livre para criar, manipular e serializar ontologias em OWL cuja versão mais recente é focada na OWL2.

A ferramenta inicia carregando os arquivos Java que contém o código-fonte do programa e utiliza o *Hibernate* como *framework* ORM. Então, com o uso do JavaParser, o programa gera a árvore de sintaxe abstrata que representa o código do programa. Esses passos são representados pela seta com o número `1` na Figura 23.

Então, a ferramenta carrega a ORM-OWL, versão operacional da ORM-O, e percorre a AST previamente gerada identificando os conceitos da ontologia presentes no código-fonte. Uma vez identificados, com o uso da OWL API os conceitos são instanciados na ontologia operacional que foi carregada em memória. O resultado desse passo, que está identificado na Figura 14 com o número `2`, é uma instância da ORM-OWL que representa o código fonte Java carregado no passo `1`.

Por fim, o último passo da ferramenta é gerar o código da ontologia instanciada na linguagem de saída utilizando o *framework* escolhido, representado na Figura 23 com o número `3`. No caso a linguagem escolhida foi Python com o uso do Django.

Para esse passo, como não foi encontrado um *parser* específico para Python na linguagem Java, a versão atual da ferramenta gera o código para o *framework* Django de forma manual. Sendo assim a ferramenta analisa as instâncias dos conceitos da ORM-O geradas no passo anterior e gera as strings de código correspondente e então imprime em um único arquivo com extensão `.py`.

A ferramenta foi testada utilizando o código Java desenvolvido para demostrar instâncias do modelo ontológico na Seção 4.2. A seguir apresentamos alguns exemplos das classes do modelo que foram utilizadas como entrada e seus respectivos códigos em Python gerados na saída.

O primeiro exemplo é o da classe `Person`. A Listagem 4.7 apresenta o código da referida classe que foi passado como entrada e a Listagem 4.8 apresenta o trecho de código do arquivo Python gerado que contém a declaração da respectiva classe.

Listagem 4.7 – Código em Java com Hibernate da classe Person

```
1 @Entity
2 @Table(name="PersinTable")
3 @Inheritance(strategy = InheritanceType.JOINED)
```

[2] https://javaparser.org/
[3] https://github.com/owlcs/owlapi

```
4 public class Person {
5
6   @Id
7   private Long id;
8   private String name;
9   @OneToOne
10  private PersonalAddress address;
11
12 }
```

Listagem 4.8 – Código em Python com Django da classe Person

```
1 class Person(models.Model):
2   address = models.OneToOneField('PersonalAddress', on_delete=models.CASCADE)
3   id = models.BigIntegerField(primary_key=True)
4   name = models.CharField(max_lenght=255)
5
6   class Meta:
7     pass
```

Outros exemplos a serem apresentados são os das classes `Professor` e `Class`. As listagens 4.9 e 4.10 apresentam, respectivamente, o código de entrada, em Java e o código de saída, em Python da classe `Professor`. Já as listagens 4.11 e 4.12 apresentam, na mesma ordem, os códigos da classe `Class`.

Listagem 4.9 – Código em Java com Hibernate da classe Professor

```
1 @Entity
2 public class Professor extends Person {
3
4   @Column(name="money")
5   private float salary;
6   @ManyToMany
7   private List<Class> professorClasses;
8
9 }
```

Listagem 4.10 – Código em Python com Django da classe Professor

```
1 class Professor(Person):
2   professorClasses = models.ManyToManyField('Class')
3   salary = models.FloatField()
4
5   class Meta:
6     db_table = 'Professor'
7     pass
```

Listagem 4.11 – Código em Java com Hibernate da classe Professor

```
1 @Entity
2 public class Class {
3
4   @Id
5   private Long id;
```

```
6     private int year;
7     @OneToMany(mappedBy="studentClass")
8     private List<Student> students;
9     @ManyToMany(mappedBy="professorClasses")
10    private List<Professor> professors;
11    @ManyToOne
12    private Institution institution;
13
14 }
```

Listagem 4.12 – Código em Python com Django da classe Professor

```
1 class Class(models.Model):
2     professors = models.ManyToManyField('Professor')
3     models.IntegerField()
4     institution = models.ForeignKey('Institution', on_delete=models.CASCADE)
5     id = models.BigIntegerField(primary_key=True)
6     year = models.IntegerField()
7
8     class Meta:
9         pass
```

O código completo da versão da ferramenta está no repositório do Github e pode ser acessado através do link <https://github.com/nemo-ufes/ORMMigrationTool/>. Também é possível encontrar neste mesmo repositório o código do exemplo Java utilizado como entrada nos testes e o código completo de saída da execução na linguagem Python, na pasta `ExampleCode`.

5 Considerações Finais

Neste trabalho é apresentada a ORM-O, uma Ontologia de Mapeamento Objeto/-Relacional. Este capítulo faz um panorama geral de todos os passos do trabalho, apresenta os pontos fortes da pesquisa e dos artefatos produzidos que aqui foram apresentados, pontua pontos negativos e também projeta trabalhos futuros para evolução e continuidade do trabalho.

Para que a proposta fosse concretizada, estudos acerca de ontologias foram realizados, incluindo a linguagem de modelagem OntoUML e a abordagem SABiO para construção de ontologias. Também foi de extrema importância o estudo e análise das ontologias reutilizadas: OOC-O, que é uma ontologia de código orientado a objeto e a RDBS-O, que é uma ontologia de sistemas de banco de dados relacionais.

Além disso foi necessária uma ampliação do conhecimento com estudos sobre a linguagem operacional OWL2 para que uma versão legível por máquinas pudesse ser produzida. Também foi feita uma análise do problema coberto pelo uso de *frameworks* ORM: a impedância objeto/relacional.

Para a proposta do modelo ontológico que cobre o escopo do uso de *frameworks* ORM no código fonte, uma aquisição de conhecimento sobre *frameworks* de linguagens OO foi necessária. Esse conhecimento foi adquirido por meio da análise das documentações do padrão JPA, Django, SQLAlchemy, ODB e QxORM, que são *frameworks* das linguagens Java, Python e C++.

Após todo esse esforço de agregação de conhecimento, foi possível desenvolver as contribuições aqui descritas, atingindo os objetivos iniciais indicados no Capítulo 1.

O objetivo de formalização de um modelo ontológico que represente o domínio do mapeamento objeto/relacional no escopo do código fonte foi atingido e o produto oriundo disso está descrito no Capítulo 3.

A avaliação do modelo foi realizada por meio da verificação de que a ontologia respondia às questões previamente apresentadas, sendo descrita na Seção 4.1. Outra etapa da avaliação foi a validação, que trouxe exemplos de instâncias do modelo ontológico em códigos reais da linguagem Java utilizando o *framework Hibernate*, detalhados na Seção 4.2. Por fim, a prova de conceito que o trabalho objetivava foi alcançada com o desenvolvimento da ferramenta de migração que utiliza a ontologia como interlíngua detalhada na Seção 4.3.

Este trabalho tem como pontos fortes a definição formal dos conceitos compartilhados nos diferentes *frameworks* ORM das diferentes linguagens de programação OO e a ferramenta de migração de código. Vale ressaltar, ainda, que este trabalho serviu

também de aplicação prática das ontologias RDBS-O (AGUIAR; FALBO; SOUZA, 2018) e OOC-O (AGUIAR; FALBO; SOUZA, 2019), contribuindo para uma avaliação positiva destas ontologias.

Algumas questões não foram abordadas, deixando a proposta com alguns pontos fracos. Como exemplo temos que, apesar do uso de diferentes documentações para aquisição de conhecimento, a validação foi feita explicitamente apenas com uma das implementações do JPA. Além disso a ferramenta desenvolvida faz apenas a migração de código em Java com Hibernate para código Python com o *framework* Django. Apesar de a ontologia ser um passo intermediário, outras combinações de migração não estão abrangidas atualmente.

Esses pontos fracos descritos podem ser solucionados. Para isso consideramos como trabalho futuro a inclusão da validação na documentação com *frameworks* de outras linguagens. Outro trabalho posterior é a adaptação da ferramenta para que seja possível migrar código de diferentes linguagens, não apenas de Java para Python. Esse passo pode ser realizado, por exemplo, substituindo o uso do JavaParser para uma ferramenta de parser de diferentes linguagens, como o ANTLR (*ANother Tool for Language Recognition*) que, a partir de uma gramática, gera um *parser* que constrói e percorre a AST.

Outro trabalho visado para uso da ontologia aqui apresentada é desenvolver ferramenta de identificação de *smells* de código. Tal objetivo é compartilhado pelas ontologias que compõem a rede SFWON.

Referências

AGUIAR, C. Z. de; FALBO, R. A.; SOUZA, V. E. S. Ontological Representation of Relational Databases. In: *Proc. of the 11th Seminar on Ontology Research in Brazil (ONTOBRAS 2018)*. São Paulo, SP, Brazil: CEUR, 2018. p. 140–151. Citado 7 vezes nas páginas 7, 8, 19, 23, 24, 25 e 54.

AGUIAR, C. Z. de; FALBO, R. A.; SOUZA, V. E. S. OOC-O: a Reference Ontology on Object-Oriented Code. In: *Submitted for publication*. [S.l.]: (under review), 2019. Citado 3 vezes nas páginas 25, 33 e 54.

AMBLER, S. W. *Mapping objects to relational databases*. 2010. <https://www.ibm.com/developerworks/library/ws-mapping-to-rdb/>. Acessado em : 05-05-2019. Citado na página 38.

BACKETT, D. *RDF 1.1 XML Syntax*. 2014. <https://www.w3.org/TR/rdf-syntax-grammar/>. Acessado em : 17-09-2019. Citado na página 28.

BACKETT, D. et al. *RDF 1.1 Turtle*. 2014. <https://www.w3.org/TR/turtle/>. Acessado em : 17-09-2019. Citado na página 29.

BAUER, C.; KING, G. *Hibernate in Action*. 1. ed. [S.l.]: Manning, 2004. ISBN 9781932394153. Citado 2 vezes nas páginas 11 e 15.

BRINGUENTE, A. C. d. O.; FALBO, R. d. A.; GUIZZARDI, G. Using a foundational ontology for reengineering a software process ontology. *Journal of Information and Data Management*, v. 2, n. 3, p. 511, 2011. Citado 2 vezes nas páginas 23 e 26.

CALERO, C. et al. An ontological approach to describe the SQL: 2003 object-relational features. *Computer Standards & Interfaces*, Elsevier, v. 28, n. 6, p. 695–713, 2006. Citado na página 30.

CAMPOS, M. L. d. A. et al. Ontologias: representando a pesquisa na área através de mapa conceitual. 2012. Citado na página 17.

CHANG, D.; IYENGAR, S. et al. Common warehouse metamodel (CWM) specification. *Object Management Group*, v. 1, 2001. Citado na página 31.

CODD, E. F. A relational model of data for large shared data banks. *Commun. ACM*, ACM, v. 13, n. 6, p. 377–387, 1970. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/362384.362685>. Citado na página 14.

DAQUIN, M.; GANGEMI, A.; HAASE, P. *Definition of ontology networks*. [S.l.]: sn, 2006. Citado na página 23.

DJANGO. *Django Documentation*. 2019. <https://docs.djangoproject.com/en/2.2/>. Acessado em : 05-05-2019. Citado na página 33.

DUARTE, B. B. et al. Ontological foundations for software requirements with a focus on requirements at runtime. *Applied Ontology*, IOS Press, v. 13, n. 2, p. 73–105, 2018. Citado 2 vezes nas páginas 23 e 26.

EVERMANN, J.; WAND, Y. Ontology based object-oriented domain modelling: fundamental concepts. *Requirements Engineering*, Springer, v. 10, n. 2, p. 146–160, 2005. Citado na página 30.

FALBO, R. A. SABiO: Systematic Approach for Building Ontologies. In: *Proc. of the 1st Joint Workshop ONTO.COM / ODISE on Ontologies in Conceptual Modeling and Information Systems Engineering*. [S.l.]: CEUR, 2014. Citado 5 vezes nas páginas 7, 20, 23, 32 e 41.

GIARETTA, P.; GUARINO, N. Ontologies and knowledge bases towards a terminological clarification. *Towards very large knowledge bases: knowledge building & knowledge sharing*, v. 25, n. 32, p. 307–317, 1995. Citado na página 17.

GRUBER, T. R. A translation approach to portable ontology specifications. *Knowledge acquisition*, Elsevier, v. 5, n. 2, p. 199–220, 1993. Citado 2 vezes nas páginas 12 e 16.

GUARINO, N. Formal ontology, conceptual analysis and knowledge representation. *International journal of human-computer studies*, Elsevier, v. 43, n. 5-6, p. 625–640, 1995. Citado na página 16.

GUARINO, N. Understanding, building and using ontologies. *International Journal of Human-Computer Studies*, Elsevier, v. 46, n. 2-3, p. 293–310, 1997. Citado na página 17.

GUARINO, N. *Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy*. [S.l.]: IOS press, 1998. v. 46. Citado 3 vezes nas páginas 7, 17 e 18.

GUARINO, N.; OBERLE, D.; STAAB, S. What is an ontology? In: *Handbook on ontologies*. [S.l.]: Springer, 2009. p. 1–17. Citado na página 17.

GUIZZARDI, G. Ontological foundations for structural conceptual models. 2005. Citado 6 vezes nas páginas 7, 16, 18, 19, 21 e 33.

GUIZZARDI, G. On ontology, ontologies, conceptualizations, modeling languages, and (meta) models. *Frontiers in artificial intelligence and applications*, IOS Press, v. 155, p. 18, 2007. Citado 2 vezes nas páginas 18 e 19.

HAWKDE, S. et al. *OWL 2 Web Ontology Language Conformance (Second Edition)*. 2012. <https://www.w3.org/TR/2012/REC-owl2-conformance-20121211/>. Acessado em : 17-09-2019. Citado na página 28.

HENDLER, J. Agents and the semantic web. *IEEE Intelligent systems*, IEEE, v. 16, n. 2, p. 30–37, 2001. Citado na página 16.

HORRIDGE, M.; PATEL-SHNEIDER, P. F. *OWL 2 Web Ontology Language Manchester Syntax (Second Edition)*. 2012. <https://www.w3.org/TR/2012/NOTE-owl2-manchester-syntax-20121211/>. Acessado em : 17-09-2019. Citado na página 29.

IRELAND, C. et al. A classification of object-relational impedance mismatch. In: *2009 First International Confernce on Advances in Databases, Knowledge, and Data Applications*. [S.l.: s.n.], 2009. p. 36–43. Citado 2 vezes nas páginas 14 e 15.

ISOTANI, S.; BITTENCOURT, I. I. *Dados Abertos Conectados: Em busca da Web do Conhecimento*. [S.l.]: Novatec Editora, 2015. Citado na página 17.

JPA. *Java Persistence API Documentation*. 2019. <https://download.oracle.com/otn-pub/jcp/persistence-2_2-mrel-spec/JavaPersistence.pdf>. Acessado em : 05-05-2019. Citado na página 33.

LABORDA, C. P. de; CONRAD, S. Relational. OWL: a data and schema representation format based on OWL. In: AUSTRALIAN COMPUTER SOCIETY, INC. *Proc. of the 2nd Asia-Pacific conference on Conceptual modelling—Volume 43*. [S.l.], 2005. p. 89–96. Citado na página 30.

MOTIK, B.; PARSIA, B.; PATEL-SHNEIDER, P. F. *OWL 2 Web Ontology Language XML Serialization (Second Edition)*. 2012. <https://www.w3.org/TR/2012/REC-owl2-xml-serialization-20121211/>. Acessado em : 17-09-2019. Citado na página 29.

MOTIK, B.; PATEL-SCHNEIDER, P. F.; GRAU, B. C. *OWL 2 Web Ontology Language Direct Semantics (Second Edition)*. 2012. <https://www.w3.org/TR/2012/REC-owl2-direct-semantics-20121211/>. Acessado em : 18-09-2019. Citado na página 29.

MOTIK, B.; PATEL-SHNEIDER, P. F.; PARSIA, B. *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)*. 2012. <https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/>. Acessado em : 17-09-2019. Citado 2 vezes nas páginas 28 e 29.

MUSEN, M. A. et al. The protégé project: a look back and a look forward. *AI matters*, NIH Public Access, v. 1, n. 4, p. 4, 2015. Citado na página 47.

Object Management Group. *OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2*. 2007. <https://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF/>. Acessado em : 18-09-2019. Citado na página 28.

ODB. *ODB Documentation*. 2019. <https://www.codesynthesis.com/products/odb/doc/manual.xhtml>. Acessado em : 05-05-2019. Citado na página 33.

ONTOUML. *Metamodelo da OntoUML*. 2019. <https://ontouml.org/ontouml/metamodel-definitions/>. Acessado em : 15-08-2019. Citado 2 vezes nas páginas 7 e 22.

OWL Working Group. *OWL 2 Web Ontology Language Document Overview (Second Edition)*. 2012. <https://www.w3.org/TR/owl2-overview/>. Acessado em : 17-09-2019. Citado 3 vezes nas páginas 7, 28 e 29.

PASTOR, O. *Diseño y Desarrollo de un Entorno de Producción Automática de Software basado en el modelo orientado a Objetos*. Tese (Doutorado) — Universitat Politècnica de València, 1992. Citado na página 30.

PATEL-SHNEIDER, P. F.; MOTIK, B. *OWL 2 Web Ontology Language Mapping to RDF Graphs (Second Edition)*. 2012. <https://www.w3.org/TR/2012/REC-owl2-mapping-to-rdf-20121211/>. Acessado em : 18-09-2019. Citado na página 28.

QXORM. *QxOrm Documentation*. 2019. <https://www.qxorm.com/qxorm_en/manual.html>. Acessado em : 05-05-2019. Citado na página 33.

RUY, F. B. et al. Seon: A software engineering ontology network. In: SPRINGER. *European Knowledge Acquisition Workshop*. [S.l.], 2016. p. 527–542. Citado 2 vezes nas páginas 23 e 26.

SCHAUB, S.; MALLOY, B. A. The Design and Evaluation of an Interoperable Translation System for Object-Oriented Software Reuse. *Journal of Object Technology*, v. 15, n. 4, p. 1–1, 2016. Citado na página 31.

SCHERP, A. et al. Designing core ontologies. *Applied Ontology*, IOS Press, v. 6, n. 3, p. 177–221, 2011. Citado 3 vezes nas páginas 7, 17 e 18.

SCHNEIDER, M. *OWL 2 Web Ontology Language RDF-Based Semantics (Second Edition)*. 2012. <https://www.w3.org/TR/2012/REC-owl2-rdf-based-semantics-20121211/>. Acessado em : 18-09-2019. Citado na página 29.

SOUZA, V. E. S.; FALBO, R. A.; GUIZZARDI, G. Designing Web Information Systems for a Framework-based Construction. In: HALPIN, T.; PROPER, E.; KROGSTIE, J. (Ed.). *Innovations in Information Systems Modeling: Methods and Best Practices*. 1. ed. [S.l.]: IGI Global, 2009. cap. 11, p. 203–237. Citado na página 15.

SQLALCHEMY. *SQLAlchemy Documentation*. 2019. <https://docs.sqlalchemy.org/en/13/>. Acessado em : 05-05-2019. Citado na página 33.

SWARTOUT, W.; TATE, A. Ontologies. *IEEE Intelligent systems and their applications*, IEEE, v. 14, n. 1, p. 18–19, 1999. Citado na página 17.

TRINH, Q.; BARKER, K.; ALHAJJ, R. RDB2ONT: A tool for generating OWL ontologies from relational database systems. In: IEEE. *Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW'06)*. [S.l.], 2006. p. 170–170. Citado na página 30.

ZANETTI, F. L.; AGUIAR, C. Z. d.; SOUZA, V. E. S. Representação Ontológica de Frameworks de Mapeamento Objeto/Relacional. In: *Proc. of the 12th Seminar on Ontology Research in Brazil (ONTOBRAS 2019)*. Porto Alegre, RS, Brasil: CEUR, 2019. Citado na página 13.