UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

ONTO-Analyst: um método extensível para a identificação e visualização de anomalias em ontologias

João Paulo Orlando

Tese de Doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

João Paulo Orlando

ONTO-Analyst: um método extensível para a identificação e visualização de anomalias em ontologias

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Dilvan de Abreu Moreira

USP – São Carlos
Outubro de 2017

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

071o Orlando, João Paulo
ONTO-Analyst: um método extensível para a
identificação e visualização de anomalias em
ontologias / João Paulo Orlando; orientador Dilvan
de Abreu Moreira. -- São Carlos, 2017.
150 p.

Tese (Doutorado - Programa de Pós-Graduação em
Ciências de Computação e Matemática Computacional) --
Instituto de Ciências Matemáticas e de Computação,
Universidade de São Paulo, 2017.

1. Web Semântica. 2. Ontologias OWL. 3.
Identificação de anomalias. 4. Visualização de
anomalias. 5. Problemas de ontologias. I. Moreira,
Dilvan de Abreu, orient. II. Título.

João Paulo Orlando

**ONTO-Analyst: an extensible method for the identification
and the visualization of anomalies in ontologies**

Doctoral dissertation submitted to the Institute of
Mathematics and Computer Sciences – ICMC-USP, in
partial fulfillment of the requirements for the degree of
the Doctorate Program in Computer Science and
Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and
Computational Mathematics

Advisor: Prof. Dr. Dilvan de Abreu Moreira

**USP – São Carlos
October 2017**

*Dedico este trabalho à minha
mais que amada família.*

AGRADECIMENTOS

Agradeço aos meus pais, pois sem eles eu não poderia existir e sempre que precisei deles, eles me ajudaram. Nem tudo o que terei em minha vida toda será o suficiente para conseguir agradecer a eles. Amo vocês Bernardete e Itacir Neri Orlando. À minha esposa Giuliana Sardi Venter, pela compreensão, motivação e auxílio nos momentos mais complicados no doutorado. Ao meu irmão Josias Marcos Orlando, pelo apoio e pelos momentos de descontração. A todos os demais familiares um grande agradecimento por tudo e, principalmente em especial ao meu primo afilhado Gustavo que tem 7 anos, desejo a ele um futuro brilhante. E espero que um dia ele esteja aqui passando por todos esses momentos em uma grande universidade.

Queria deixar um agradecimento especial ao meu orientador Prof. Dr. Dilvan de Abreu Moreira, pela confiança e por toda a ajuda dedicada ao meu trabalho. E ainda agradeço a todos demais docentes do ICMC - USP e da URI que contribuíram para minha formação.

Aos amigos do ICMC e colegas de trabalho, especialmente a Flor Karina Mamani e Kleberson Serique que me ajudaram várias vezes no desenvolvimento deste doutorado. Aos colegas de república que compartilharam dos mesmos desafios acadêmicos: Bruno Guazzelli Batista, Bruno Tardiole Kuehne, Daniel Fernando Pigatto, João Paulo Franco Eustáquio, Leonardo Santos Mesquita, Paulo Henrique Moreira Gurgel e Paulo Sérgio Franco Eustáquio.

E por fim, mas não menos importante, a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro, ao Instituto de Ciências Matemáticas e de Computação e à Universidade de São Paulo pela estrutura disponibilizada para realização deste projeto.

*“It matters not how strait the gate,
How charged with punishments the scroll,
I am the master of my fate,
I am the captain of my soul. ”*
(William Ernest Henley - Invictus)

RESUMO

ORLANDO, J. P. **ONTO-Analyst: um método extensível para a identificação e visualização de anomalias em ontologias**. 2017. 150 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2017.

A Web Semântica é uma extensão da Web em que as informações tem um significado explícito, permitindo que computadores e pessoas trabalhem em cooperação. Para definir os significados explicitamente, são usadas ontologias na estruturação das informações. À medida que mais campos científicos adotam tecnologias da Web Semântica, mais ontologias complexas são necessárias. Além disso, a garantia de qualidade das ontologias e seu gerenciamento ficam prejudicados quanto mais essas ontologias aumentam em tamanho e complexidade. Uma das causas para essas dificuldades é a existência de problemas, também chamados de anomalias, na estrutura das ontologias. Essas anomalias englobam desde problemas sutis, como conceitos mal projetados, até erros mais graves, como inconsistências. A identificação e a eliminação de anomalias podem diminuir o tamanho da ontologia e tornar sua compreensão mais fácil. Contudo, métodos para identificar anomalias encontrados na literatura não visualizam anomalias, muitos não trabalham com OWL e não são extensíveis por usuários. Por essas razões, um novo método para identificar e visualizar anomalias em ontologias, o ONTO-Analyst, foi criado. Ele permite aos desenvolvedores identificar automaticamente anomalias, usando consultas SPARQL, e visualizá-las em forma de grafos. Esse método usa uma ontologia proposta, a *META-data description For Ontologies/Rules* (MetaFOR), para descrever a estrutura de outras ontologias, e consultas SPARQL para identificar anomalias nessa descrição. Uma vez identificadas, as anomalias podem ser apresentadas na forma de grafos. Um protótipo de sistema, chamado ONTO-Analyst, foi criado para a validação desse método e testado em um conjunto representativo de ontologias, por meio da verificação de anomalias representativas. O protótipo testou 18 tipos de anomalias retirados da literatura científica, em um conjunto de 608 ontologias OWL de 4 repositórios públicos importantes e dois artigos. O sistema detectou 4,4 milhões de ocorrências de anomalias nas 608 ontologias: 3,5 milhões de ocorrências de um mesmo tipo e 900 mil distribuídas em 11 outros tipos. Essas anomalias ocorreram em várias partes das ontologias, como classes, propriedades de objetos e de dados, etc. Num segundo teste foi realizado um estudo de caso das visualizações geradas pelo protótipo ONTO-Analyst das anomalias encontradas no primeiro teste. Visualizações de 11 tipos diferentes de anomalias foram automaticamente geradas. O protótipo mostrou que cada visualização apresentava os elementos envolvidos na anomalia e que pelo menos uma solução podia ser deduzida a partir da visualização. Esses resultados demonstram que o método pode eficientemente encontrar ocorrências de anomalias

em um conjunto representativo de ontologias OWL, e que as visualizações facilitam o entendimento e correção da anomalia encontrada. Para estender os tipos de anomalias detectáveis, usuários podem escrever novas consultas SPARQL.

Palavras-chave: Web Semântica, Ontologias OWL, Identificação de Anomalias, Visualização de Anomalias, Problemas de Ontologias.

ABSTRACT

ORLANDO, J. P. **ONTO-Analyst: an extensible method for the identification and the visualization of anomalies in ontologies**. 2017. 150 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2017.

The Semantic Web is an extension of the World Wide Web in which the information has explicit meaning, allowing computers and people to work in cooperation. In order to explicitly define meaning, ontologies are used to structure information. As more scientific fields adopt Semantic Web technologies, more complex ontologies are needed. Moreover, the quality assurance of the ontologies and their management are undermined as these ontologies increase in size and complexity. One of the causes for these difficulties is the existence of problems, also called anomalies, in the ontologies' structure. These anomalies range from subtle problems, such as poorly projected concepts, to more serious ones, such as inconsistencies. The identification and elimination of anomalies can diminish the ontologies' size and provide a better understanding of the ontologies. However, methods to identify anomalies found in the literature do not provide anomaly visualizations, many do not work on OWL ontologies or are not user extensible. For these reasons, a new method for anomaly identification and visualization, the ONTO-Analyst, was created. It allows ontology developers to automatically identify anomalies, using SPARQL queries, and visualize them as graph images. The method uses a proposed ontology, the METAdata description For Ontologies/Rules (MetaFOR), to describe the structure of other ontologies, and SPARQL queries to identify anomalies in this description. Once identified, the anomalies can be presented as graph images. A system prototype, the ONTO-Analyst, was created in order to validate this method and it was tested in a representative set of ontologies, through the verification of representative anomalies. The prototype tested 18 types of anomalies, taken from the scientific literature, in a set of 608 OWL ontologies from major public repositories and two articles. The system detected 4.4 million anomaly occurrences in the 608 ontologies: 3.5 million occurrences from the same type and 900 thousand distributed in 11 other types. These anomalies occurred in various parts of the ontologies, such as classes, object and data properties, etc. In a second test, a case study was performed in the visualizations generated by the ONTO-Analyst prototype, from the anomalies found in the first test. It was shown that each visualization presented the elements involved in the anomaly and that at least one possible solution could be deduced from the visualization. These results demonstrate that the method can efficiently find anomaly occurrences in a representative set of OWL ontologies and that the visualization aids in the understanding and correcting of said anomalies. In order to extend the types of detectable anomalies, users can write new SPARQL queries.

Keywords: Semantic Web, OWL Ontologies, Anomaly Identification, Anomaly Visualization, Ontology Problems.

LISTA DE ILUSTRAÇÕES

Figura 1 – Exemplo de Grafo RDF	38
Figura 2 – Regra SWRL com a divisão entre antecedente e consequente	44
Figura 3 – Regra SWRL destacando um átomo com o seu predicado e suas variáveis	45
Figura 4 – Exemplo de uma ocorrência da anomalia <i>Lonely disjoint classes</i>	61
Figura 5 – Exemplo da anomalia <i>Partition error in taxonomy</i>	62
Figura 6 – Exemplo de uma ocorrência da anomalia <i>Exact circularity in taxonomy</i>	62
Figura 7 – Exemplo de uma ocorrência da anomalia <i>Redundancy by repetitive taxonomic definition</i>	63
Figura 8 – Exemplo de uma ocorrência da anomalia <i>Circular properties</i>	64
Figura 9 – Exemplo de uma ontologia visualizada em forma de árvore no layout da esquerda para direita na ferramenta OntoGraf	73
Figura 10 – Exemplo de uma ontologia visualizada em forma de grafo direcionado pela força na ferramenta WebVOWL	74
Figura 11 – Exemplo de visualização de uma taxonomia em lista indentada	75
Figura 12 – Exemplo de visualização da categoria Zoomable	77
Figura 13 – Visualização da Gene Ontology (GO) no Treemap	78
Figura 14 – Fluxograma mostrando o processo do método ONTO-Analyst.	82
Figura 15 – A propriedade de dados <code>has_current_age</code> (esquerda, em verde) X sua representação no formato MetaFOR (direita, em laranja).	85
Figura 16 – Representação no formato MetaFOR da propriedade <code>has_adoptive_daughter</code> e suas relações com outras entidades da ontologia original.	87
Figura 17 – Representação de propriedades de dados de entidades da ontologia analisada no formato MetaFOR.	88
Figura 18 – Tela principal do protótipo ONTO-Analyst: Ele tem um painel editor para consultas SPARQL, uma tabela para listar resultados das consultas e um painel para a representação das visualizações	89
Figura 19 – Diagrama de componentes do ONTO-Analyst com suas interfaces.	90
Figura 20 – Representação gráfica de uma ocorrência de anomalia onde as instâncias MetaFOR são nós e as arestas são relações entre as instâncias.	104
Figura 21 – Tipos de nós disponíveis para a visualização das ontologias no sistema ONTO-Analyst. Cada nó é um retângulo com um símbolo para representar um tipo de entidade analisada.	105

Figura 22 – Tipos de arestas para a visualização das ontologias no sistema ONTO-Analyst. Cada aresta é uma linha com um símbolo que representa o tipo de relacionamento entre os nós	105
Figura 23 – Influência da origem do repositório nos tipos de anomalias.	116
Figura 24 – Influência da ocorrência de anomalias por tipo de entidade de cada repositório.	116
Figura 25 – Relação da porcentagem de ocorrência da anomalia <i>Redundancy by repetitive taxonomic definition</i> com o total de classes e propriedades de cada ontologia (cada ponto representa uma ontologia).	118
Figura 26 – Exemplo de uma ocorrência da anomalia <i>Lonely disjoint class</i> na ontologia GRO.	119
Figura 27 – Distribuição da porcentagem de ocorrência da anomalia <i>Lazy class or property</i> com o total de classes e propriedades de cada ontologia	120
Figura 28 – Relação entre a ocorrência da anomalia <i>Lazy class or property</i> e o total de classes/propriedades em cada ontologia.	121
Figura 29 – Ocorrência da anomalia <i>Circular properties</i> na ontologia ICF. No lado esquerdo está a definição da propriedade <code>primaryCareParent</code> . No lado direito está a definição da propriedade <code>primaryCareSubclasses</code> . Cada propriedade é inversa da outra, mas ambas tem o mesmo <code>rdfs:domain</code> e <code>rdfs:range</code>	121
Figura 30 – Visualização de uma ocorrência da anomalia <i>Lonely disjoint classes</i> na ontologia GFVO.	126
Figura 31 – Visualização de uma ocorrência da anomalia <i>Partition error in taxonomy</i> na ontologia DIAGONT.	127
Figura 32 – Visualização de uma ocorrência da anomalia <i>Exact circularity in taxonomy</i> na ontologia QIBO.	128
Figura 33 – Visualização de uma ocorrência da anomalia <i>Redundancy by repetitive taxonomic definition</i> na ontologia SWEET.	129
Figura 34 – Visualização de uma ocorrência da anomalia <i>Lazy class or property</i> na ontologia FHHO.	130
Figura 35 – Visualização de uma ocorrência da anomalia <i>Chains of inheritance</i> na ontologia FHHO.	131
Figura 36 – Visualização de uma ocorrência da anomalia <i>Circular Properties</i> na ontologia ICF.	131
Figura 37 – Visualização de uma ocorrência da anomalia <i>Circularity between rules and taxonomy</i> na ontologia <i>UnderwaterEnvironment</i>	132
Figura 38 – Visualização de uma ocorrência da anomalia <i>Redundant Derivation in the Antecedent</i> na ontologia <i>ContextOntology</i>	133

Figura 39 – Visualização de uma ocorrência da anomalia <i>Redundant implication of transitivity or symmetry</i> na ontologia <i>Rules for processing graph knowledgebase</i>	134
Figura 40 – Visualização de uma ocorrência da anomalia <i>Rule subsumption</i> na ontologia elaborada por Young <i>et al.</i> (2009).	135

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – Exemplo de dados em RDF	47
Código-fonte 2 – Exemplo de consulta sobre dados em RDF	47
Código-fonte 3 – Identificação da redundância <i>Redundant derivation in the antecedent</i> com o Datalog*	52
Código-fonte 4 – Exemplo de uma ocorrência da anomalia <i>Circularity between rules and taxonomy</i>	65
Código-fonte 5 – Exemplo de uma ocorrência da anomalia <i>Contradicting rules</i> . .	66
Código-fonte 6 – Exemplo de uma ocorrência da anomalia <i>Redundant derivation in the antecedent</i>	66
Código-fonte 7 – Exemplo de uma ocorrência da anomalia <i>Redundant implication</i> .	67
Código-fonte 8 – Exemplos de duas ocorrências da anomalia <i>Incompatible rule antecedent</i>	68
Código-fonte 9 – Exemplos de duas ocorrências da anomalia <i>Self-contradicting rule</i>	68
Código-fonte 10 – Exemplos de duas ocorrências da anomalia <i>Rule subsumption</i> . .	69
Código-fonte 11 – Prefixos padrões usados nas consultas SPARQL que identificam anomalias	93
Código-fonte 12 – Identificação da anomalia <i>Chains of inheritance</i> usando SPARQL	93
Código-fonte 13 – Identificação da anomalia <i>Lonely disjoint classes</i> usando SPARQL	94
Código-fonte 14 – Identificação da anomalia <i>Partition error in taxonomy</i> usando SPARQL	94
Código-fonte 15 – Identificação da anomalia <i>Exact circularity in taxonomy</i> usando SPARQL	95
Código-fonte 16 – Identificação da anomalia <i>Redundancy by repetitive taxonomic definition</i> usando SPARQL	95
Código-fonte 17 – Identificação da anomalia <i>Lazy class or property</i> usando SPARQL	95
Código-fonte 18 – Identificação da anomalia <i>Circular properties</i> usando SPARQL .	96
Código-fonte 19 – Identificação da anomalia <i>Multiple functional properties</i> usando SPARQL	96
Código-fonte 20 – Identificação do primeiro caso da anomalia <i>Redundant cardinalities</i> usando SPARQL	97
Código-fonte 21 – Identificação do segundo caso da anomalia <i>Redundant cardinalities</i> usando SPARQL	97

Código-fonte 22 – Identificação da anomalia <i>Circularity between rules and taxonomy</i> usando SPARQL	97
Código-fonte 23 – Identificação da anomalia <i>Contradicting rules</i> usando SPARQL .	98
Código-fonte 24 – Identificação da anomalia <i>Incompatible rule antecedent</i> em classes usando SPARQL	98
Código-fonte 25 – Identificação da anomalia <i>Incompatible rule antecedent</i> em propriedades de objetos usando SPARQL	99
Código-fonte 26 – Identificação da anomalia <i>Redundant derivation in the antecedent</i> usando SPARQL	99
Código-fonte 27 – Identificação da anomalia <i>Redundant implication</i> usando SPARQL	99
Código-fonte 28 – Identificação da anomalia <i>Redundant implication of transitivity</i> com propriedades transitivas usando SPARQL	100
Código-fonte 29 – Identificação da anomalia <i>Redundant implication of symmetry</i> com propriedades simétricas usando SPARQL	101
Código-fonte 30 – Identificação da anomalia <i>Redundant use of transitivity</i> com propriedades transitivas usando SPARQL	101
Código-fonte 31 – Identificação da anomalia <i>Redundant use of symmetry</i> com propriedades simétricas usando SPARQL	102
Código-fonte 32 – Identificação da anomalia <i>Self-contradicting rule</i> em classes usando SPARQL	102
Código-fonte 33 – Identificação da anomalia <i>Self-contradicting rule</i> em propriedades de objetos usando SPARQL	103
Código-fonte 34 – Identificação da anomalia <i>Rule subsumption</i> usando SPARQL . .	103
Código-fonte 35 – Regra da ontologia BCGO com a anomalia <i>Circularity between rules and taxonomy</i>	122
Código-fonte 36 – Regra da ontologia contextOntology com a anomalia <i>Redundant derivation in the antecedent</i>	122
Código-fonte 37 – Regra com a anomalia <i>Redundant implication of transitivity or symmetry</i>	123
Código-fonte 38 – Regra com a anomalia <i>Rule subsumption</i>	123

LISTA DE TABELAS

Tabela 1 – Resultado de uma consulta SPARQL	47
Tabela 2 – Limitações encontradas nos trabalhos relacionados	58
Tabela 3 – Lista das anomalias escolhidas. Incluindo nome, entidade(s) afetada(s) e tipo.	60
Tabela 4 – Repositórios com o número de ontologias e suas datas de download. . .	110
Tabela 5 – Resultados gerais do processo de conversão das ontologias analisadas para o formato MetaFOR	110
Tabela 6 – Ontologias convertidas para o formato MetaFOR que falharam na va- lidação pelos reasoners.	111
Tabela 7 – Ontologias descartadas com problemas no processo de conversão para o formato MetaFOR	112
Tabela 8 – Resumo das estruturas nas 608 ontologias analisadas	112
Tabela 9 – Resultados gerais das identificações de anomalias que ocorrem em clas- ses/propriedades	113
Tabela 10 – Resultados gerais das identificações de anomalias que ocorrem em pro- priedades de objetos	114
Tabela 11 – Resultados gerais das identificações de anomalias que ocorrem em re- gras	115
Tabela 12 – Resultados das ontologias mais relevantes do Bioportal.	125

LISTA DE ABREVIATURAS E SIGLAS

BCGO	<i>Breast Cancer Grading Ontology</i>
BNO	<i>Bionutrition Ontology</i>
DIAGONT	<i>Diagnostic Ontology</i>
FBbi	<i>Biological Imaging Methods Ontology</i>
FHHO	<i>Family Health History Ontology</i>
FMA	<i>Fundamental Model of Anatomy</i>
GFVO	<i>Genomic Feature and Variation Ontology</i>
GO	<i>Gene Ontology</i>
GRO	<i>Gene Regulation Ontology</i>
ICF	<i>International Classification of Functioning, Disability and Health</i>
MetaFOR	<i>METAdata description For Ontologies/Rules</i>
NCBO	<i>National Center for Biomedical Ontology</i>
NIFSTD	<i>Neuroscience Information Framework Standard Ontology</i>
ODP	<i>Ontology Design Patterns</i>
OWL	<i>Web Ontology Language</i>
QIBO	<i>Quantitative Imaging Biomarker Ontology</i>
RDF	<i>Resource Description Framework</i>
RDFS	<i>Resource Description Framework Schema</i>
RIF	<i>Rule Interchange Format</i>
SPARQL	<i>Simple Protocol and RDF Query Language</i>
SWEET	<i>Semantic Web for Earth and Environment Technology Ontology</i>
SWO	<i>Software Ontology</i>
SWRL	<i>Semantic Web Rule Language</i>
URI	<i>Uniform Resource Identifier</i>
W3C	<i>World Wide Web Consortium</i>
XML	<i>eXtensible Markup Language</i>

SUMÁRIO

1	INTRODUÇÃO	29
1.1	Objetivo	31
1.2	Contribuições ao Estado da arte	32
1.3	Organização da Tese	33
2	WEB SEMÂNTICA	35
2.1	Ontologias	36
2.2	Padrões para definir ontologias	37
2.2.1	<i>XML, RDF e RDFS</i>	37
2.2.2	<i>OWL</i>	38
2.3	Padrão para definição de regras em ontologias	44
2.4	Padrão para consultas em ontologias	46
2.5	Considerações finais	47
3	IDENTIFICAÇÃO DE ANOMALIAS EM ONTOLOGIAS	49
3.1	Tipos de Anomalias	50
3.2	Trabalhos relacionados com a identificação de anomalias	51
3.2.1	<i>Anomalies in ontologies with rules</i>	52
3.2.2	<i>Ontology Evaluation: a pitfall-based approach to ontology diagnosis</i>	53
3.2.3	<i>Discovering Anomalies in Semantic Web Rules</i>	54
3.2.4	<i>A Method of Eliminating Direct Super-Class to Detect Cycle of Concepts</i>	54
3.2.5	<i>Managing and Refining Rule Sets for SWRL</i>	55
3.2.6	<i>Elimination of Redundancy in Ontologies</i>	55
3.2.7	<i>OntoCheck: verifying ontology naming conventions and metadata completeness in Protégé 4</i>	56
3.2.8	<i>XD Analyzer</i>	56
3.2.9	<i>Evaluation of Taxonomic Knowledge in Ontologies and Knowledge Bases</i>	57
3.3	Resumo das limitações dos trabalhos relacionados	57
3.4	Conjunto de anomalias escolhidas da literatura científica	59
3.4.1	<i>Chains of inheritance</i>	59
3.4.2	<i>Lonely disjoint classes</i>	60

3.4.3	<i>Partition error in taxonomy</i>	61
3.4.4	<i>Exact circularity in taxonomy</i>	62
3.4.5	<i>Redundancy by repetitive taxonomic definition</i>	62
3.4.6	<i>Lazy class or property</i>	63
3.4.7	<i>Circular properties</i>	64
3.4.8	<i>Multiple functional properties</i>	64
3.4.9	<i>Redundant cardinalities</i>	64
3.4.10	<i>Circularity between rules and taxonomy</i>	65
3.4.11	<i>Contradicting rules</i>	65
3.4.12	<i>Redundant derivation in the antecedent</i>	66
3.4.13	<i>Redundant implication</i>	66
3.4.14	<i>Redundant implication of transitivity or symmetry</i>	67
3.4.15	<i>Redundant use of transitivity and symmetry</i>	67
3.4.16	<i>Incompatible rule antecedent</i>	67
3.4.17	<i>Self-contradicting rule</i>	68
3.4.18	<i>Rule subsumption</i>	68
3.5	<i>Considerações finais</i>	69
4	VISUALIZAÇÃO DA INFORMAÇÃO APLICADA À ONTOLOGIAS	71
4.1	<i>Node-link and tree</i>	72
4.2	<i>Indented list</i>	75
4.3	<i>Zoomable</i>	76
4.4	<i>Space-filling</i>	76
4.5	<i>Focus + context or distortion</i>	77
4.6	<i>3D Information landscapes</i>	79
4.7	<i>Discussão da aplicação das categorias de visualização para anomalias</i>	79
4.8	<i>Considerações finais</i>	80
5	MÉTODO ONTO-ANALYST PARA IDENTIFICAÇÃO E VISUALIZAÇÃO DE ANOMALIAS	81
5.1	<i>Ontologia MetaFOR</i>	83
5.1.1	<i>Representação de entidades</i>	84
5.1.2	<i>Representações de relações entre entidades</i>	86
5.1.3	<i>Representações de relações entre entidades e valores</i>	87
5.1.4	<i>Extensibilidade</i>	88
5.2	<i>Protótipo do sistema ONTO-Analyst</i>	89
5.2.1	<i>Estágio 1: Preparação da conversão da ontologia analisada para o formato MetaFOR</i>	90
5.2.2	<i>Estágio 2: Conversão da ontologia analisada para o formato MetaFOR</i>	92

5.2.3	<i>Estágio 3: Operações com uma ontologia no formato MetaFOR</i>	92
5.3	Consultas SPARQL para identificação das anomalias no formato MetaFOR	93
5.3.1	<i>Anomalias que ocorrem em classes e propriedades</i>	93
5.3.2	<i>Anomalias que ocorrem em cardinalidades</i>	96
5.3.3	<i>Anomalias que ocorrem em regras SWRL</i>	97
5.4	Visualização das anomalias	104
5.4.1	<i>Criação dos grafos</i>	106
5.5	Considerações finais	106
6	EXPERIMENTOS	107
6.1	Análise quantitativa da ocorrência de anomalias em ontologias públicas	107
6.1.1	<i>Seleção dos repositórios de ontologias</i>	108
6.1.2	<i>Download das ontologias</i>	109
6.1.3	<i>Conversão para o formato MetaFOR</i>	109
6.1.3.1	<i>Resumo das estruturas convertidas</i>	111
6.1.4	<i>Identificação e análise das anomalias</i>	112
6.1.4.1	<i>Visão geral dos resultados</i>	112
6.1.4.2	<i>Análise de algumas anomalias específicas</i>	115
6.1.4.2.1	Exact circularity in taxonomy	116
6.1.4.2.2	Partition error in taxonomy	117
6.1.4.2.3	Redundancy by repetitive taxonomic definition	118
6.1.4.2.4	Lonely disjoint class	119
6.1.4.2.5	Lazy class or property	120
6.1.4.2.6	Circular properties	121
6.1.4.2.7	Circularity between rules and taxonomy	122
6.1.4.2.8	Redundant derivation in the antecedent	122
6.1.4.2.9	Redundant implication of transitivity or symmetry	123
6.1.4.2.10	Rule subsumption	123
6.1.4.3	<i>Ontologias mais relevantes</i>	124
6.2	Estudo de caso com as visualizações de anomalias encontradas nas ontologias analisadas	126
6.2.1	<i>Lonely disjoint classes</i>	126
6.2.2	<i>Partition error in taxonomy</i>	126
6.2.3	<i>Exact circularity in taxonomy</i>	127
6.2.4	<i>Redundancy by repetitive taxonomic definition</i>	128
6.2.5	<i>Lazy class or property</i>	129
6.2.6	<i>Chains of inheritance</i>	130
6.2.7	<i>Circular properties</i>	130

6.2.8	<i>Circularity between rules and taxonomy</i>	132
6.2.9	<i>Redundant derivation in the antecedent</i>	132
6.2.10	<i>Redundant implication of transitivity or symmetry</i>	133
6.2.11	<i>Rule subsumption</i>	134
6.3	Considerações finais	134
7	CONCLUSÕES	137
7.1	Contribuições	139
7.2	Trabalhos futuros	140
REFERÊNCIAS		143

INTRODUÇÃO

A Web Semântica é uma maneira de representar informações de uma forma mais significativa para que máquinas possam processar essas informações. Para que essas informações possam ser processadas por máquinas, a Web Semântica explora a associação de significados explícitos aos conteúdos de documentos presentes na Web ([BERNERS-LEE; HENDLER; LASSILA, 2001](#)).

Para explicitar esses significados, é necessário estruturar as informações usando ontologias. Na ciência da computação, o termo ontologia refere-se a uma especificação formal e explícita de uma conceituação compartilhada ([STUDER; BENJAMINS; FENSEL, 1998](#)). Formal refere-se ao fato de que a ontologia deve ser legível por máquinas. Explícito significa que os conceitos utilizados e as restrições sobre seu uso são explicitamente definidos. Conceituação se refere a um modelo abstrato de algum fenômeno no mundo, identificando os conceitos relevantes desse fenômeno. E, por último, compartilhado refere-se à noção de que uma ontologia captura o conhecimento consensual, isto é, não é privado de algum indivíduo, mas aceito por um grupo.

Para definir e instanciar uma ontologia na Web, o *World Wide Web Consortium* (W3C) recomenda o uso da *Web Ontology Language* (OWL). OWL é uma linguagem de marcação para publicar e compartilhar ontologias na Web ([BECHHOFFER et al., 2004](#)). Nem sempre a linguagem OWL é suficiente para os desenvolvedores ou usuários de ontologias, por isso existem outras linguagens que complementam as definições de uma ontologia ou podem servir para extrair informações de uma ontologia. Em especial, nesta tese, as linguagens *Semantic Web Rule Language* (SWRL) e *Simple Protocol and RDF Query Language* (SPARQL) são necessárias e, respectivamente, são utilizadas para a escrita de regras (dedução de conhecimento) e de consultas (recuperação de conhecimento) em uma ontologia.

O objetivo da Web Semântica é criar uma internet em que os computadores pos-

sam entender e navegar nas informações da Web. Tornar isso realidade vai requerer uma grande quantidade de ontologias complexas que, quando alinhadas (integradas) entre si, formarão uma rede de relações semânticas (SEIDENBERG; RECTOR, 2006). À medida que todo esse conhecimento aumenta de tamanho, fica mais difícil entender, manter, modificar e classificar essas ontologias (PAK; ZHOU, 2009). Uma das causas para essas dificuldades é a existência de problemas, também chamados de anomalias, na estrutura das ontologias. Podem existir diversos tipos de anomalias, desde problemas sutis, como partes mal projetadas na ontologia, até erros graves, como inconsistências.

Identificar e eliminar todos esses tipos de anomalias, em uma ontologia, pode tornar mais fácil a manutenção, entendimento, e gerenciamento da mesma. Pensando nisso, vários trabalhos da literatura científica apresentam tentativas de auxiliar a identificação de anomalias em ontologias. Ao fazer uma revisão da literatura científica, foi possível identificar importantes limitações nos trabalhos encontrados:

1. Dois dos trabalhos não identificam anomalias em ontologias OWL;
2. Seis dos trabalhos não identificam anomalias em regras SWRL;
3. Oito dos trabalhos não permitem que o conjunto de anomalias identificadas possa ser estendido por desenvolvedores de ontologias;
4. Nenhum dos trabalhos permite que desenvolvedores modifiquem ou estendam a lista de anomalias identificadas por meio de linguagens bem conhecidas pela comunidade da Web Semântica;
5. Nenhum dos trabalhos permite visualizar graficamente as anomalias identificadas em ontologias;
6. Nenhum dos trabalhos realiza análises quantitativas da real ocorrência de anomalias em grandes conjuntos de ontologias encontrados em repositórios públicos.

A eficácia das ferramentas da Web Semântica depende da qualidade das ontologias usadas (PAK; ZHOU, 2009). Um dos pontos críticos para manter essa qualidade é identificar e eliminar anomalias em ontologias. Em diversas áreas em que ontologias já são bastante usadas, como por exemplo a área das ciências da vida, existem diversas ontologias chave, como a *Gene Ontology* (GO) ou *Fundamental Model of Anatomy* (FMA), que são grandes e desenvolvidas por grandes grupos de pesquisadores distribuídos por vários países. Isso complica ainda mais a tarefa de manter a qualidade da ontologia e sua aderência às melhores práticas de projeto. Outro problema é que vários tipos de anomalias não são verificados pelos *reasoners*, como redundâncias, circularidades ou deficiências. De um modo geral, a identificação e eliminação desses tipos de anomalias pode melhorar a compreensão e diminuir o tamanho da ontologia.

Por outro lado, existem repositórios onde novas ontologias são disponibilizadas de maneira aberta, como o BioPortal, para uso em diferentes projetos e ferramentas. Algumas dessas ontologias são bem conhecidas, mas muitas não o são e por isso não são reusadas. Reuso de ontologias é algo muito importante e estimulado pela comunidade, mas como ter uma ideia da qualidade de uma ontologia nova? Uma ferramenta que pudesse identificar e listar automaticamente as anomalias presentes numa ontologia pode dar uma ideia mais clara da qualidade/maturidade da mesma, dependendo dos tipos e quantidades das anomalias encontradas.

Um método e ferramentas baseadas nele, que permitisse a detecção automática de conjuntos representativos de anomalias, seria de grande ajuda para resolver muitos dos problemas mencionados. Seria vantajoso também que ele permitisse a visualização das anomalias, como forma de ajudar na correção das mesmas, e que permitisse, de maneira relativamente fácil, a codificação de novas identificações de anomalias.

1.1 Objetivo

O objetivo principal desta tese é propor e validar um método para identificar e visualizar ocorrências de anomalias em ontologias OWL, chamado ONTO-Analyst. Ele usa uma ontologia, a *METADATA description For Ontologies/Rules* (MetaFOR), para representar a estrutura de outras ontologias, e consultas SPARQL, para identificar anomalias nessas representações. Ele também permite a visualização automática das anomalias encontradas.

Ontologias OWL e a linguagem SPARQL são tecnologias padrões na Web Semântica amplamente conhecidas pela comunidade. Isso permite que desenvolvedores de ontologias possam entender como as anomalias são detectadas e, se necessário, modificar as consultas SPARQL utilizadas ou mesmo criar novas consultas para novos tipos de anomalias. É possível usar também regras na linguagem SWRL no lugar de consultas SPARQL para a identificação de anomalias.

Entre as vantagens do método ONTO-Analyst está a visualização automática das anomalias na forma de grafos, com nós representando as entidades envolvidas na anomalia e as arestas representando as relações entre elas.

Como objetivos secundários, esta tese teve a implementação de uma lista inicial de anomalias representativas no método proposto e um estudo da real ocorrência dessas anomalias em um conjunto grande de ontologias representativas (das ontologias OWL comumente usadas em projetos atualmente).

Para alcançar esses objetivos, foram cumpridas as seguintes etapas básicas:

- Elaboração de um protótipo do sistema de identificação das anomalias (Usando a

MetaFOR e consultas SPARQL);

- Experimentos com esse sistema de identificação num conjunto de 608 ontologias públicas;
- Expansão do protótipo do sistema para incluir a visualização de anomalias;
- Experimentos com a visualizações de 11 tipos diferentes de anomalias.

Os resultados dos experimentos mostraram que o método, implementado no protótipo (chamado de sistema ONTO-Analyst), foi capaz de identificar de maneira automática 12 anomalias de 18 diferentes tipos em 608 ontologias. Eles também mostram que a geração automática de visualizações dos 11 exemplos de anomalias (de 11 tipos diferentes) foi capaz de mostrar os elementos envolvidos diretamente e indiretamente com as anomalias.

1.2 Contribuições ao Estado da arte

Os objetivos foram cumpridos e as seguintes contribuições científicas e tecnológicas foram geradas por esta tese:

- Elaboração da ontologia MetaFOR usada para representar a estrutura de ontologias analisadas pelo método, permitindo que desenvolvedores de ontologias possam usar a linguagem SPARQL para realizar a identificação de anomalias;
- Implementação de um conjunto de consultas SPARQL que, quando executadas em ontologias no formato MetaFOR, identificam as ocorrências de um conjunto de 18 tipos de anomalias. Essas consultas permitem que as identificações sejam extensíveis/modificáveis. Usuários de ontologias podem alterar as identificações existentes ou inserir novas com a criação de novas consultas SPARQL (ou novas regras SWRL);
- Implementação do sistema ONTO-Analyst (protótipo), desenvolvido na linguagem Java, de código aberto, disponível na web, que implementa o método ONTOAnalyst. Ele permite a adição de novas estruturas, não previstas inicialmente na ontologia MetaFOR por meio da implementação de interfaces java, e novas consultas SPARQL. Ele também permite a visualização das ocorrências de anomalias encontradas para facilitar seu entendimento;
- Análise quantitativa da real ocorrência de um conjunto representativo de tipos de anomalias em 608 ontologias públicas e apresentação de quais tipos são mais comuns. No total, foram escolhidos da literatura 18 tipos diferentes de anomalias. Desses, 12 tipos foram encontrados nas ontologias analisadas, provando que muitas anomalias ainda ocorrem com frequência;

- Estudo de caso sobre visualizações de 11 tipos de anomalias que ocorrem em ontologias públicas, mostrando que a visualização permite identificar facilmente muitas das possíveis soluções das anomalias.

1.3 Organização da Tese

Os demais capítulos desta tese estão organizados como segue. No Capítulo 2 são apresentados conceitos sobre Web Semântica, ontologias e linguagens padrões para definição de ontologias, definição de regras e consultas em ontologias.

No Capítulo 3 é apresentada a área de identificação de anomalias em ontologias. Ainda nesse mesmo capítulo, são mostrados os trabalhos que apresentam esforços na identificação de anomalias e uma lista de anomalias selecionadas para posterior implementação no método proposto por esta tese.

No Capítulo 4 são apresentados os diferentes tipos de técnicas implementados em trabalhos que apresentam esforços na visualização de ontologias. No Capítulo 5 são apresentadas as implementações da ontologia MetaFOR e do protótipo ONTO-Analyst que são utilizados para identificar e visualizar as anomalias.

No Capítulo 6 são apresentados os experimentos realizados com a MetaFOR e o ONTO-Analyst em ontologias públicas. Os experimentos são divididos em uma análise quantitativa das ocorrências de anomalias em ontologias públicas e um estudo de caso das visualizações geradas pelo ONTO-Analyst para essas ocorrências de anomalias.

WEB SEMÂNTICA

Desde seu surgimento, a *World Wide Web* (ou simplesmente Web) vem mudando o modo como as pessoas se comunicam e trocam informações. A Web cresceu rapidamente, tornando-se a maior fonte de informações da humanidade ([SUGUMARAN; GULLA, 2011](#)). Muito do sucesso da Web se deve aos motores de busca baseados em palavras-chave, tais como Google e Yahoo. No entanto, podemos citar graves problemas associados a utilização de motores de busca ([ANTONIOU; HARMELEN, 2008](#)):

- *Recall* alto, precisão baixa: mesmo que os documentos mais relevantes sejam recuperados, eles ficam em meio a diversos documentos pouco relevantes ou irrelevantes.
- Baixo ou nenhum *recall*: às vezes, não se obtém nenhuma resposta relevante para a busca realizada. Esse é um problema menos frequente nos atuais motores de busca.
- Resultados altamente sensíveis ao vocabulário: os resultados são muito dependentes de vocabulário, pois motores de busca são fortemente baseados em palavras-chave.
- Os resultados são documentos Web individuais: não raro, as informações estão espalhadas por vários documentos. Às vezes, a solução desse problema se dá por meio de várias consultas para coletar todos os documentos pertinentes e, em seguida, da extração manual da informação necessária, colocando-a em conjunto.

Observando esses problemas, percebe-se que eles estão associados à dificuldade que os computadores têm de entender o significado dos conteúdos presentes na Web. É evidente que existem ferramentas que podem recuperar documentos, dividi-los em partes, verificar a ortografia e contar suas palavras. Entretanto, quando se trata de interpretar frases e extrair informações úteis para os usuários, a capacidade dos softwares atuais ainda é muito limitada, pois o significado ainda é algo muito difícil de ser distinguido ([ANTONIOU; HARMELEN, 2008](#)). Uma possível solução é a representação semântica dos

conteúdos Web. Essa representação fornece a formalidade necessária para que máquinas possam processar os significados desses conteúdos.

A Web Semântica é a solução proposta pelo W3C para fornecer essa representação semântica. O principal responsável por essa ideia é Tim Berners-Lee, o próprio inventor da Web. De acordo com [Berners-Lee, Hendler e Lassila \(2001\)](#), a Web Semântica é uma rede de dados que podem ser processados diretamente ou indiretamente por máquinas. Para possibilitar esse processamento, os computadores necessitam ter acesso a coleções estruturadas de informações (dados e metadados) e a conjuntos de regras de inferência sobre esses conteúdos, para que seja possível o processamento automatizado. Para representar essas informações, é utilizado o conceito de ontologia (que será apresentado na Seção 2.1). Porém, como só ontologias não seriam suficientes para dar semântica à Web, é preciso também a adoção de tecnologias padrões para representar ontologias, adicionar regras e recuperar informações. Esses padrões serão apresentados nas Seções 2.2, 2.3 e 2.4.

2.1 Ontologias

As ontologias são vistas como a base para a construção da Web Semântica ([HEFLIN, 2004](#)). Segundo [Gruber \(1993\)](#), esse termo originalmente foi emprestado da filosofia, onde significa um relato sistemático da existência. Em ciência da computação, esse termo representa uma especificação formal e explícita de uma conceituação compartilhada ([STUDER; BENJAMINS; FENSEL, 1998](#)). Conceituação se refere a um modelo abstrato de algum fenômeno no mundo, identificando os conceitos relevantes deste fenômeno. Explícito significa que os conceitos utilizados e as restrições sobre seu uso são explicitamente definidos. Formal refere-se ao fato de que a ontologia deve ser legível pelas máquinas. Compartilhado refere-se à noção de que uma ontologia captura o conhecimento consensual, isto é, não é privado de algum indivíduo, mas aceito por um grupo. Tal compreensão compartilhada é necessária para superar as diferenças de terminologia. Essas diferenças podem ser superadas por meio do mapeamento da terminologia específica, para uma ontologia compartilhada, ou por meio da definição de mapeamentos diretos entre as ontologias. Em ambos os casos, é fácil perceber que as ontologias apoiam a interoperabilidade semântica.

Ontologias são amplamente utilizadas para diferentes fins e em diferentes comunidades, mas a sua função de descrever formalmente um domínio e seus componentes básicos (recursos) se mantém a mesma. Os componentes básicos de uma ontologia são classes, relações, axiomas e instâncias ([NOY; MCGUINNESS, 2001](#)). As classes são organizadas em forma de taxonomia e denotam conceitos importantes do domínio. As relações (propriedades) representam os tipos de interações entre os conceitos de um domínio. Elas descrevem várias características e atributos de uma classe ou instância. Os axiomas têm

a função de modelar afirmações sobre as classes ou instâncias, ou seja, são regras assumidas como verdadeiras no domínio em questão. Por último, as instâncias são utilizadas para representar elementos específicos, ou seja, os próprios dados que, juntamente com a definição de uma ontologia, constituem a base de conhecimento (NOY; MCGUINNESS, 2001).

2.2 Padrões para definir ontologias

Para Berners-Lee, Hendler e Lassila (2001), só ontologias não seriam suficientes para imprimir semântica à Web. É preciso também a adoção de tecnologias padrões para representar e recuperar informações em ontologias. Um dos principais responsáveis pela definição de padrões para criação e uso de ontologias na Web é o W3C. O primeiro padrão W3C para ontologias que será apresentado é o *Resource Description Framework* (RDF), que está disponível desde 1999 e é uma especialização da *eXtensible Markup Language* (XML). O RDF usa triplas para descrever recursos e relações entre eles. Ele é uma linguagem padrão para ontologias simples e é usado em muitas aplicações existentes como, por exemplo, para o gerenciamento de conteúdos, bibliotecas digitais e *e-commerce* (SUGUMARAN; GULLA, 2011). Para complementar o RDF, foi lançada a OWL (versão 1.0) como recomendação do W3C em 2004. A OWL trouxe uma maior interpretabilidade de conteúdos da Web por máquinas quando comparada com o uso do RDF (SUGUMARAN; GULLA, 2011). Nesta subseção, todos esses padrões W3C para definição de ontologias e seus componentes serão apresentados.

2.2.1 XML, RDF e RDFS

A XML é uma metalinguagem universal para a definição de marcação. Ela fornece uma estrutura uniforme e um conjunto de ferramentas, como analisadores para intercâmbio de dados e metadados entre aplicações. No entanto, a XML não fornece qualquer meio para o entendimento da semântica, ou seja, do significado dos dados.

Visando sanar essa dificuldade de fornecer semântica, o RDF foi criado. Ele nada mais é do que uma especialização do XML, um esquema para a definição de informações na Web. Ele fornece meios para expressar o significado de termos e conceitos de modo que os computadores possam processá-los (BERNERS-LEE; HENDLER; LASSILA, 2001). Para expressar termos e conceitos, o RDF usa o conceito de *Uniform Resource Identifier* (URI) para nomear os recursos de uma tripla. Cada tripla é composta por três recursos: um sujeito, um predicado e um objeto (KLYNE; CARROLL, 2004). Com esse modelo simples, o RDF permite que os dados estruturados e semiestruturados possam ser misturados, expostos e compartilhados entre aplicações diferentes (KLYNE; CARROLL, 2004).

A estrutura básica de qualquer expressão em RDF é uma coleção de triplas que

Figura 1 – Exemplo de Grafo RDF

Fonte: Elaborada pelo autor.

representam afirmações. Isso pode ser ilustrado por um diagrama de nós e arcos dirigidos no qual cada tripla é representada como uma ligação nó-arco-nó. Utilizando a representação de grafos (Figura 1), os nós RDF são o sujeito e o objeto. O arco, ligando os dois (direcionado para o objeto) é o predicado. Cada nó pode ser uma URI ou ter o seu valor “em branco” (não podendo ser identificado). Nós objeto também podem ser literais (strings, números, etc.) (KLYNE; CARROLL, 2004). Os predicados são identificados por URIs e podem ser também nós em um grafo.

Uma das limitações do RDF é a falta de representação de hierarquias de classes ou de propriedades. Ela foi superada com a criação do *Resource Description Framework Schema* (RDFS) que adicionou as definições `rdfs:subClassOf` e `rdfs:subPropertyOf`. Além disso, o RDFS estendeu algumas definições para elementos RDF, tais como `rdfs:domain` e `rdfs:range` (explicados na subseção 2.2.2).

Para Feigenbaum *et al.* (2007), o RDFS é o bloco mais fundamental de construção para a Web Semântica, pois, além de poder ser utilizado para criar dados semânticos, é também utilizado como base para as linguagens para representação de ontologias da Web Semântica.

2.2.2 OWL

Para Staab, Maedche e Handschuh (2001), os dados em RDFS são fracamente interligados, de modo que a Web Semântica necessita de técnicas mais sofisticadas para que não existam ambiguidades na representação. Desta forma, o *Web Ontology Working Group*¹ identificou várias características para Web Semântica que requeriam mais expressividade que os padrões RDF e RDFS podiam oferecer (ANTONIOU; HARMELEN, 2008). Vários pesquisadores dos Estados Unidos e da Europa definiram uma linguagem mais rica

¹ <https://www.w3.org/2001/sw/WebOnt/>

chamada DAML+OIL² e essa foi o ponto de partida para a definição da OWL.

A OWL é o padrão recomendado pelo W3C, desde 10 de fevereiro de 2004, para publicar e compartilhar ontologias na Web (BECHHOFFER *et al.*, 2004). Essa linguagem foi projetada para aplicações que necessitam processar o conteúdo da informação, ao invés de apenas apresentar informações em nós como RDF(S) (MCGUINNESS; HARMELEN, 2004). OWL é uma linguagem que permite que uma informação semântica seja explicitamente associada ao conteúdo de dados e formalmente especificada por meio de ontologias compartilhadas na Web.

Em sua primeira versão, a OWL possuía três sub-linguagens, projetadas para comunidades específicas:

- A OWL *Lite* tinha como alvo usuários que necessitam principalmente de uma classificação hierárquica e restrições simples (MCGUINNESS; HARMELEN, 2004). Por exemplo, embora suporte restrições de cardinalidade, ela só permite que a cardinalidade assuma valores de 0 ou 1.
- A OWL DL foi feita para usuários que queriam a máxima expressividade, enquanto mantinham a computabilidade (todas as conclusões tem a garantia de ser computáveis) e decidibilidade (todas as computações terminam em tempo finito) (MCGUINNESS; HARMELEN, 2004). Embora a OWL DL inclua todas as construções da linguagem OWL, elas podem ser usadas somente sob certas restrições. Por exemplo, enquanto uma classe pode ser subclasse de muitas classes, uma classe não pode ser uma instância de outra classe. OWL DL é assim chamada devido à sua correspondência com as lógicas de descrição (*description logic*), um campo de pesquisa que estuda as lógicas que formam a base formal da OWL (MCGUINNESS; HARMELEN, 2004).
- A OWL *Full* foi direcionada para usuários que queriam a máxima expressividade e a liberdade sintática do RDF(S) sem nenhuma garantia computacional, como as mencionadas anteriormente (MCGUINNESS; HARMELEN, 2004). É importante ressaltar, entretanto, que é improvável que algum software de raciocínio (*reasoner*) seja capaz de suportar completamente cada recurso da OWL *Full* (MCGUINNESS; HARMELEN, 2004).

Posteriormente, foram agregados novos recursos à linguagem OWL, dando origem a uma nova recomendação, chamada OWL 2, que é uma extensão e revisão da primeira versão da OWL (MOTIK *et al.*, 2012). A OWL 2 adiciona novas funcionalidades em relação a sua primeira versão. Algumas dessas novas funções não passam de pequenos detalhes, como a união disjunta de classes. Outras, porém, oferecem nova expressividade,

² <http://www.daml.org/2001/03/daml+oil-index.html>

como chaves, redes de propriedades, tipos de dados mais ricos, intervalos de dados, restrições de cardinalidade qualificada e propriedades assimétricas e reflexivas (MOTIK *et al.*, 2012).

A OWL 2 também contém sub-linguagens conhecidas como *profiles*. Os *profiles* são versões restritas da OWL 2, que têm algum ganho específico para o processo de classificação e inferência, além de contribuírem para a resolução de problemas relacionados com escalabilidade e eficiência (MOTIK *et al.*, 2012). Cada *profile* é definido pelas restrições da OWL DL. Os *profiles* são:

- OWL 2 EL é particularmente útil em aplicações que empregam ontologias que contém um grande número de propriedades e/ou classes (MOTIK *et al.*, 2012). Esse perfil é um subconjunto do OWL 2 em que os problemas de raciocínio podem ser processados em tempo polinomial em relação ao tamanho da ontologia (MOTIK *et al.*, 2012).
- OWL 2 QL é voltado para aplicações que utilizam grandes volumes de dados instanciados e têm necessidade de fazer consultas sobre esses dados. Esse *profile* é recomendado para quando a resposta da consulta é a tarefa mais importante (MOTIK *et al.*, 2012). Em OWL 2 QL, para se responder a consultas, podem ser usados sistemas convencionais de bancos de dados relacionais (MOTIK *et al.*, 2012).
- OWL 2 RL é voltado para aplicações que exigem raciocínio escalável sem sacrificar muito a força expressiva (MOTIK *et al.*, 2012). Ele é projetado para os casos em que é necessária muita expressividade sem perder a eficiência (MOTIK *et al.*, 2012). Esse perfil foi projetado para ser implementável por meio de *resoners* baseados em regras. O raciocínio com o perfil OWL 2 RL é polinomial em relação ao tamanho da ontologia.

Elementos de uma ontologia

Como foi visto, existem vários *profiles* da OWL, porém todos eles tem elementos semelhantes na estrutura de suas ontologias. Na maioria dos casos, os principais elementos de uma ontologia OWL são classes, propriedades, instâncias de classes (ou indivíduos de classes), e relações entre instâncias de classes (SMITH; WELTY; MCGUINNESS, 2004).

Os elementos mais básicos de uma ontologia devem corresponder a classes, que fazem parte de árvores taxonômicas, e instâncias, pertencentes a, pelo menos, uma classe. A classe raiz de toda hierarquia é `OWL:Thing`, por isso, toda instância pertence a classe `OWL:Thing`.

É importante destacar como é feita a distinção entre uma classe e uma instância em OWL. Uma classe é simplesmente um nome que contém uma coleção de propriedades

(características) que descrevem um conjunto de instâncias. Já as instâncias são os membros desses conjuntos. Assim, as classes devem corresponder naturalmente a ocorrências de um conjunto de Coisas (*Thing*) de um domínio e as instâncias devem corresponder as entidades reais que podem ser agrupadas nessas classes (SMITH; WELTY; MCGUINNESS, 2004).

Na construção de ontologias, essa distinção é frequentemente confundida podendo ser sempre interpretada de duas formas. Em determinados contextos algo que é, obviamente, uma classe pode ser considerado como uma instância. Por exemplo, em uma ontologia de vinhos em que temos o conceito **Grape**, que se destina a indicar o conjunto de todas as variedades de Uva. **CabernetSauvignonGrape** poderia ser uma instância dessa classe **Grape**, já que denota a variedade de uva Cabernet Sauvignon. Entretanto, **CabernetSauvignonGrape** poderia ser considerada como uma classe, pois poderia representar o conjunto de todas as uvas Cabernet Sauvignon. Isso mostra que a definição das classes e suas instâncias dependerá do domínio em questão.

Somente existindo classes e instâncias, as ontologias seriam pouco interessante, por isso existem propriedades que servem para afirmar fatos sobre todas as instâncias de uma classe e/ou fatos específicos sobre determinada instância. Em OWL existem dois tipos de propriedades (ANTONIOU; HARMELEN, 2008):

- *Object properties*: que relacionam instâncias a outras instâncias.
- *Data properties*: que relacionam instâncias a dados.

Além de definir propriedades, é possível definir algumas características dessas propriedades que fornecem um poderoso mecanismo no raciocínio sobre elas. Abaixo, estão algumas das definições que foram importantes para esta tese:

- *Transitive Property*: se definimos que uma propriedade P é transitiva, isso irá significar que, a partir das relações (x, y) e (y, z) dessa propriedade, é inferido a relação (x, z) .
- *Symmetric Property*: se definimos que uma propriedade P é simétrica, isso irá significar que a partir da relação (x, y) dessa propriedade é inferido a relação (y, x) .
- *Functional Property*: uma propriedade funcional pode ter apenas um único valor de y para cada instância x , ou seja, não podem existir dois valores distintos y_1 e y_2 para uma instância x de uma propriedade funcional (BECHHOFFER *et al.*, 2004).
- *Inverse Functional Property*: se for afirmado que uma propriedade P é uma *Inverse Functional Property*, então se afirma que um valor y só pode ser o valor P para uma única instância x , ou seja, não poderá haver duas instâncias x_1 e x_2 , em que os pares (x_1, y) e (x_2, y) são ocorrência de P (BECHHOFFER *et al.*, 2004).

- *Inverse Of*: uma propriedade pode ser definida como inversa de outra. Toda a vez que ocorrer uma relação (x, y) com uma propriedade que tem uma propriedade inversa, será inferido (y, x) com a propriedade inversa.

Em adição à declaração de características das propriedades, é possível limitar uma propriedade a um *domain* (`rdfs:domain`) e também a um *range* (`rdfs:range`) e, por consequência, podemos definir restrições na definição de um *domain* e de um *range*. Um *domain* define que uma propriedade só pode ocorrer em determinadas instâncias pertencentes a uma ou mais classes. Já o *range* determina quais são as instâncias alvo daquela *object property* ou os valores alvo para uma *data property*.

Em OWL, classes são definidas através de axiomas. Eles podem incluir várias restrições sobre as propriedades que os membros da classe podem ter. As possíveis restrições que podem ser adicionadas ao *domain* e/ou *range* dessas propriedades são:

- *allValuesFrom* é usada para requerer que, cada instância de uma classe que use a propriedade indicada, só possa usar, como valores para essa propriedade, instâncias da classe indicada na restrição.
- *someValuesFrom* é usada para requerer que, cada instância de uma classe, use, pelo menos uma vez, a propriedade indicada com uma instância da classe indicada na restrição.
- *minCardinality* é usada para descrever o número mínimo de ocorrências que aquela propriedade deverá ter nas instâncias da classe.
- *maxCardinality* é usado para descrever o máximo de ocorrências que aquela propriedade deverá ter nas instâncias da classe.
- *exactCardinality* é usado para descrever a quantidade exata de ocorrências que aquela propriedade deverá ter nas instâncias da classe.

Outros axiomas, que uma ontologia pode conter, são os mapeamentos entre elementos (classes, propriedades, etc) de uma ou mais ontologias. A sua principal função é facilitar que ontologias possam ser compartilhadas, reutilizadas e integradas com outras. É importante perceber que grande parte do esforço no desenvolvimento de uma ontologia é dedicado a se ligar os conjuntos classes e propriedades de forma a maximizar implicações. É sempre esperado realizar afirmações simples que gerem implicações amplas e úteis (SMITH; WELTY; MCGUINNESS, 2004). Alguns dos mapeamentos que auxiliam na integração de ontologias e que são essenciais para esta tese são os seguintes:

- *equivalentClass*, *equivalentProperty*: eles definem que duas classes ou duas propriedades são sinônimas.

- *disjointWith*: ele firma que, dada duas classes disjuntas, as instâncias que pertencem a uma classe da disjunção não podem pertencer a outra.

Avaliação e qualidade de uma ontologia

As linguagens padrão W3C de definição de ontologias não garantem a qualidade da ontologia criada. A área de avaliação de ontologias é necessária para melhorar a qualidade das ontologias. A avaliação é realizada por meio de um julgamento técnico de ontologias, seus ambientes de software associados e sua documentação durante cada fase e entre as fases do seu ciclo de vida (GÓMEZ-PÉREZ; JURISTO; PAZOS, 1995). Essa atividade é dividida em avaliação de (GÓMEZ-PÉREZ; JURISTO; PAZOS, 1995):

- Cada definição individual e axioma;
- Coleções de definições e axiomas que são explícitos nas definições da ontologia;
- Definições que são importadas de outras ontologias;
- Axiomas que podem ser inferidos usando coleções de definições e axiomas.

O termo avaliação de ontologias engloba os termos verificação e validação (POVEDA-VILLALÓN, 2016). O termo verificação de ontologias refere-se à garantir que as definições da ontologia satisfaçam corretamente seus requisitos, suas questões de competência ou funcionem corretamente no mundo real (GÓMEZ-PÉREZ; JURISTO; PAZOS, 1995). Para garantir que uma ontologia seja bem verificada, é necessário verificar sua arquitetura, sua sintaxe e seu conteúdo.

O termo validação de ontologias refere-se à validação do significado das definições encontradas na ontologia analisada (GÓMEZ-PÉREZ; JURISTO; PAZOS, 1995). Esse significado deve representar o mundo real para o qual ele foi criado.

Segundo Poveda-Villalón (2016), surgiram inúmeras abordagens para a avaliação ontológica classificadas em:

- *Frameworks* genéricos de avaliação geral da qualidade de uma ontologia;
- Diretrizes específicas para avaliar a adequação ontológica e a consistência lógica das taxonomias;
- Avaliação uma ontologia em termos de reutilização e uso final dos conceitos dela;
- Modelos de qualidade baseados em características, critérios e métricas;
- Métodos para avaliação baseada em padrões desejados ou indesejados.

Cada um desses tipos de abordagens apresenta diferentes maneiras de garantir a qualidade de uma ontologia, mas de um modo geral a qualidade de uma ontologia pode ser definida pelas seguintes características [Gómez-Pérez, Juristo e Pazos \(1995\)](#): consistência, completude, concisão, expansibilidade e sensibilidade.

Inconsistência refere-se à obtenção de conhecimento contraditório a partir de uma entrada de definição válida. Incompletude refere-se à falta de definições nas representações de uma ontologia. Não concisão refere-se à ocorrência de definições desnecessárias. Inexpansibilidade refere-se ao esforço necessário em adicionar novas definições que alterem definições existentes e corretas. Sensibilidade relaciona-se a como as pequenas mudanças em uma definição alteram o conjunto de definições bem definidas que já estão validadas.

Embora a definição de qualidade seja em grande parte um trabalho do desenvolvedor da ontologia com o uso de abordagens pré-definidas, a validação da qualidade verificará as características mostradas acima.

2.3 Padrão para definição de regras em ontologias

A expressividade da OWL nem sempre é suficiente para modelar todos os tipos de problemas. Especificamente, ela não tem suporte a cláusulas no formato de *Horn*, que representam sentenças de implicação ([HORROCKS et al., 2004](#)). Em lógica, a implicação é a indicação do tipo "SE...ENTÃO...", indicando que uma condição deve ser satisfeita necessariamente para que a outra seja verdadeira. Para suprir essa deficiência, foi proposta a linguagem SWRL para o W3C em 2004. Até o momento, a SWRL não virou uma recomendação do W3C, mas ela é uma linguagem popular que conta com o suporte de muitas ferramentas, tais como Protégé, Pellet e Hermit ([ORLANDO; MUSEN; MOREIRA, 2015](#)). A SWRL permite incluir regras que podem inferir novos conhecimentos em determinadas instâncias de ontologias OWL ([HORROCKS et al., 2004](#)).

Figura 2 – Regra SWRL com a divisão entre antecedente e consequente

Fonte: Adaptada de [Orlando, Rívoli e Moreira \(2012\)](#).

Regras, em SWRL, são divididas em duas partes (Figura 2): o antecedente e o consequente. Cada regra é uma implicação entre o antecedente e o consequente (se o antecedente for verdadeiro, então o consequente deve ser inferido). Ambas as partes consistem em uma conjunção de um conjunto de átomos. Os átomos, por sua vez, são compostos por

um predicado (que pode ser uma classe, uma propriedade, uma igualdade, uma diferença, um tipo de dado ou uma função) e um ou mais argumentos (Figura 3). Os argumentos podem ser referências a instâncias, valores ou variáveis (conjunto de valores ou instâncias). Os átomos podem ser classificados, com base no seu tipo de predicado, em seis tipos diferentes (HASSANPOUR; O’CONNOR; DAS, 2009). Eles são apresentados a seguir conforme a descrição de Hassanpour, O’Connor e Das (2009):

- Predicado do tipo *class*: ele corresponde a uma classe definida na ontologia e recebe um argumento. O argumento deve ser uma instância ou uma variável para instâncias.
- Predicado do tipo *object property*: ele corresponde a uma propriedade definida na ontologia e recebe dois argumentos. Os argumentos devem ser instâncias ou variáveis para instâncias.
- Predicado do tipo *datavalued property*: ele corresponde a uma propriedade definida na ontologia e recebe dois argumentos. O primeiro deve ser uma instância ou uma variável para instâncias e o segundo é um valor ou variável para valores.
- Predicado do tipo *data range*: ele corresponde a um tipo de dado definido na ontologia e recebe um valor como argumento.
- Predicado do tipo *same* ou *different*: ele corresponde ao termo “*same as*” ou “*different from*” e recebe duas instâncias como argumentos, definindo que elas devem ser iguais ou diferentes respectivamente.
- Predicado do tipo *built-in*: ele corresponde a um conjunto de funções pré-definidas (comparações, funções matemáticas, lógicas, ...) ou definidas pelo usuário que recebem um ou mais argumentos e retornam verdadeiro quando esses satisfazem o predicado.

Figura 3 – Regra SWRL destacando um átomo com o seu predicado e suas variáveis

Fonte: Elaborada pelo autor.

Embora as regras SWRL possam ser representadas em mais de um formato, o formato de leitura humano é adotado na Figura 2. Nesse formato, uma seta é usada para

separar antecedente e conseqüente, o acento circunflexo representa a conjunção entre os átomos e o sinal de interrogação distingue as variáveis dos nomes de instâncias.

Semanticamente, o exemplo da Figura 2 ilustra que se o pai/mãe de um indivíduo possui um irmão, então esse irmão é tio do indivíduo em questão. A regra possui três átomos (todos do tipo *object property*) sendo dois no antecedente e um no conseqüente (cujos predicados são `hasParent`, `hasBrother` e `hasUncle`, respectivamente) e faz uso de três variáveis para instâncias: `?x1`, `?x2` e `?x3`.

Cabe ressaltar que existem outras formas de representar regras na Web Semântica, entres elas o OWL 2 RL (já apresentado anteriormente) e o *Rule Interchange Format* (RIF). O OWL 2 RL pode representar parte das regras SWRL, mas ainda é necessário usar SWRL como complemento para se obter uma maior expressividade (RAKIB; FARUQI; MACCAULL, 2012). Por exemplo, os *built-ins* não estão disponíveis em OWL 2 RL. Já o RIF não é um padrão de escrita de regras e sim um padrão recomendado pelo W3C para intercâmbio entre diversos formatos de regras (KIFER, 2008). O mais importante é destacar que o RIF tem suporte a OWL 2 RL e a SWRL, ou seja, ambos são intercambiáveis. Então, mesmo que o trabalho desenvolvido nesta tese use o padrão de regras SWRL é possível, posteriormente, que o mesmo trabalho seja realizado com OWL 2 RL.

2.4 Padrão para consultas em ontologias

A SPARQL é uma linguagem para consulta de dados RDF, aceita como recomendação pelo W3C em 2008. Cabe ressaltar que, como o OWL é uma extensão do RDF, dados em OWL também podem ser consultados com essa linguagem.

A sintaxe da linguagem SPARQL é semelhante à linguagem de consulta de banco de dados SQL, pois algumas cláusulas são as mesmas (*select* e *where*). A cláusula *select* contém as variáveis que serão retornadas da consulta e a cláusula *where* contém o filtro da consulta. Conceitualmente, as consultas realizadas em SPARQL usam padrões representados em grafos RDF (triplas), mas podem conter nomes de variáveis no lugar de alguns nós (recursos) ou links (propriedades) (PRUD'HOMMEAUX; SEABORNE, 2008).

Os Códigos-fonte 1 e 2 e a Tabela 1 mostram, respectivamente, um conjunto de dados RDF, uma consulta SPARQL e os dados retornados pela consulta. No Código-fonte 1 estão sendo definidas quatro triplas RDF (informações): os títulos e preços dos livros um e dois. Já o Código-fonte 2, mostra uma consulta SPARQL que seleciona o Título e o Preço de cada livro (cláusula *select*). A cláusula *where*, no mesmo código-fonte, filtra as informações que irão aparecer na Tabela 1. A única restrição em *where* é que os livros retornados tenham um preço inferior a 30,50. Então, na tabela de resultado, é retornado apenas o Livro 1 (book1). Além disso, como pode ser visto na consulta, SPARQL permite usar os prefixos para abreviar as URIs em namespaces e esses podem ser usados em toda

a consulta, por exemplo, a URI: `<http://example.org/ns#>` para o namespace `ns`.

Código-fonte 1 – Exemplo de dados em RDF

```
1: @prefix dc: <http://purl.org/dc/elements/1.1/> .
2: @prefix : <http://example.org/book/> .
3: @prefix ns: <http://example.org/ns#> .
4:
5: :book1 dc:title "SPARQL Tutorial" .
6: :book1 ns:price 42 .
7: :book2 dc:title "The Semantic Web" .
8: :book2 ns:price 23 .
```

Fonte: [Prud'hommeaux e Seaborne \(2008\)](#).

Código-fonte 2 – Exemplo de consulta sobre dados em RDF

```
1: PREFIX dc: <http://purl.org/dc/elements/1.1/>
2: PREFIX ns: <http://example.org/ns#>
3:
4: SELECT ?title ?price
5: WHERE { ?x ns:price ?price .
6:   FILTER (?price < 30.5)
7:   ?x dc:title ?title .}
```

Fonte: [Prud'hommeaux e Seaborne \(2008\)](#).

Tabela 1 – Resultado de um consulta SPARQL

title	price
"The Semantic Web"	23

Fonte: [Prud'hommeaux e Seaborne \(2008\)](#).

2.5 Considerações finais

Ao longo deste capítulo foram apresentados os conceitos relacionados à Web Semântica que são considerados mais importantes para o desenvolvimento dos objetivos desta tese. A Web Semântica permite associar significado aos conteúdos presentes na Web, de forma que esses conteúdos podem ser processados por máquinas. Para realizar esse processamento é necessário ter uma coleção estruturada de informações (baseada em ontologias) e um conjunto de regras de inferência (além de padrões para a definição dessas ontologias e regras).

No próximo capítulo serão tratados os problemas que podem ocorrer no desenvolvimento dessas ontologias e regras, bem como os métodos já criados para realizar a identificação desses problemas.

IDENTIFICAÇÃO DE ANOMALIAS EM ONTOLOGIAS

As ontologias são o alicerce da Web Semântica. Elas definem os conceitos e os relacionamentos que possibilitam a interoperabilidade entre sistemas. O conhecimento capturado em ontologias pode ser usado, entre outras coisas, para anotar dados, distinguir entre homônimos¹ e polissemia², generalizar ou especializar conceitos e inferir informações inteiramente novas (implícitas) (SEIDENBERG; RECTOR, 2006).

O objetivo da Web Semântica é criar uma internet em que os computadores possam entender e navegar nas informações da *World Wide Web*. Tornar isso realidade requererá uma grande quantidade de ontologias de domínio que, quando alinhadas entre si, formam uma rede de relações semânticas (SEIDENBERG; RECTOR, 2006). À medida que todo esse conhecimento aumenta de tamanho, fica mais difícil adicionar, usar, entender, manter, modificar e classificar essas informações. Muitas vezes, essas dificuldades podem ser devidas a problemas como erros, redundâncias, circularidades, inconsistências ou deficiências nas definições encontradas em uma ontologia.

Vários autores (BAUMEISTER; SEIPEL, 2010; ZHANG, 2005; FAHAD; QADIR, 2008; PAK; ZHOU, 2009) definem esses tipos de problemas, encontrados em ontologias, como anomalias. Outras nomenclaturas também podem ser encontrados na literatura, como *Pitfalls*, *Problems*, *Partition Errors*, etc. Neste trabalho, o termo anomalia será adotado para se referir a todos esses tipos de problemas.

¹ Homônimo é a relação entre duas ou mais palavras que, apesar de possuírem significados diferentes, possuem a mesma estrutura fonológica.

² Polissemia é a propriedade que uma mesma palavra tem de apresentar vários significados.

3.1 Tipos de Anomalias

Anomalias podem ter diferentes consequências em uma ontologia, isso dependerá do tipo da anomalia. Para mostrar as possíveis consequências das anomalias em ontologias, a classificação de tipos de anomalias apresentada por [Baumeister e Seipel \(2010\)](#) foi utilizada. Essa classificação é adotada porque ela engloba desde anomalias simples, como áreas mal projetadas, até outras que podem causar problemas sérios em uma ontologia. A classificação de [Baumeister e Seipel \(2010\)](#) divide as anomalias em circularidades, redundâncias, inconsistências e deficiências:

Circularidades podem ocorrer quando ciclos são formados em uma ontologia. Esse ciclos ocorrem, essencialmente, quando uma classe é definida como uma especialização ou generalização dela mesma ([GÓMEZ-PÉREZ, 1999](#); [BAUMEISTER; SEIPEL, 2010](#)). A principal consequência desses ciclos na ontologia é a piora no desempenho dos *reasoners* ([BAUMEISTER; SEIPEL, 2010](#)). Como é um ciclo, cada vez que o *reasoner* infere algo pertencente a uma entidade do ciclo, o ciclo deve ser processado novamente pelo mesmo.

Redundâncias são definições repetidas na ontologia ou a existência de uma afirmação que pode ser inferida por meio de outras afirmações ([GÓMEZ-PÉREZ, 1999](#)). Eliminar essas redundância pode otimizar o armazenamento da ontologia e o desempenho dos *reasoners* ([GRIMM; WISSMANN, 2011](#)). Além disso, a existência de uma redundância pode apontar falhas no projeto da ontologia ([ORLANDO; MUSEN; MOREIRA, 2015](#)).

Inconsistências são conhecimentos contraditórios ([BAUMEISTER; SEIPEL, 2010](#)). Esse tipo de problema pode acontecer de duas formas: na primeira, uma afirmação inconsistente gera a ocorrência de um erro dentro da ontologia e, na segunda, uma afirmação inconsistente nunca será disparada, pois a contradição é o pré-requisito para realizar uma inferência, ou seja, essa inferência nunca será realizada. No primeiro caso, os *reasoners* conseguem identificar a anomalia facilmente, pois, quando forem inferindo novos conhecimentos na ontologia, eles verificam se esses conhecimentos cumprem as regras preestabelecidas na ontologia. Porém, no segundo caso, as inferências de novos conhecimentos nunca serão realizadas e, conseqüentemente, os *reasoners*, como FaCT++, Pellet e HermiT³, não identificam essas situações.

Deficiências são problemas mais sutis em relação aos outros porque são considerados problemas que afetam o projeto de uma ontologia ([BAUMEISTER; SEIPEL, 2010](#)). Esse tipo de anomalia não causa erros na inferência de conhecimento, mas é um

³ Os *reasoners* FaCT++, Pellet e HermiT foram testados com essas situações e não foram capazes de identificar esse tipo de inconsistência

indicador de áreas problemáticas e mal projetadas em uma ontologia (FAHAD; QADIR; SHAH, 2008). A remoção desse tipo de problema pode melhorar a usabilidade e a manutenção de uma ontologia (FAHAD; QADIR, 2008).

Identificar todos esses diferentes tipos de anomalias pode melhorar a qualidade das ontologias e, conseqüentemente, fazer com que a Web Semântica alcance seu objetivo final. Pensando nisso, vários trabalhos apresentam soluções para identificar anomalias em ontologias.

Neste capítulo, na Seção 3.2 é apresentada uma seleção de trabalhos relacionados com a identificação de anomalias em ontologias. Essa seleção seguiu aos seguintes critérios:

- Trabalhos mais recentes que identificam qualquer uma dessas 4 classificações de Baumeister e Seipel (2010) em ontologias OWL e/ou em regras SWRL;
- Trabalhos que apresentam listas de anomalias a serem identificadas em ontologias.

Após a apresentação dos trabalhos relacionados, na Seção 3.3 é apresentado um resumo das limitações encontradas na identificação de anomalias que esta tese pretende resolver. Além disso, na Seção 3.4 é apresentada uma seleção de anomalias encontradas nos trabalhos relacionados que servirão para demonstrar o método de identificação de anomalias proposto por esta tese. As anomalias escolhidas são um grupo representativo de anomalias distribuídas entre as 4 classificações de Baumeister e Seipel (2010) e entre ocorrências em diversas partes de uma ontologia (classes, propriedades, regras e cardinalidades).

3.2 Trabalhos relacionados com a identificação de anomalias

Cada uma das seguintes subseções apresentará um trabalho relacionado ou um conjunto de trabalhos relacionados com a identificação de anomalias em ontologias. Essas ontologias podem conter ou não regras em SWRL. Em cada trabalho, será apresentada a descrição, a validação (quando se aplicar) e as limitações do trabalho. Todos os itens apresentados nos trabalhos relacionados servirão como base para a construção do método proposto por esta tese.

Uma limitação importante encontrada em todos os trabalhos apresentados é que nenhum deles possui uma visualização gráfica das anomalias identificadas. Focar na apresentação das anomalias pode facilitar o entendimento da anomalia e, conseqüentemente facilitar a correção da mesma.

3.2.1 Anomalies in ontologies with rules

Baumeister e Seipel (2010) apresentam uma linguagem chamada Datalog* para identificar anomalias em ontologias com regras. A Datalog* é uma extensão da linguagem Datalog que pode ser utilizada no sistema DisLog Developers' Kit (DDK), também desenvolvido pelos autores (BAUMEISTER; SEIPEL, 2010). Datalog é uma linguagem não procedural de consulta, baseada em Prolog, em que podem ser escritas regras que realizam essas identificações. Os autores usam essa extensão do Datalog para escrever regras que identificam as ocorrências de anomalias em ontologias. Para fazer isso, eles adicionaram os predicados, que fazem parte da identificação de anomalias, ao Datalog*.

Um exemplo da linguagem Datalog* é mostrado abaixo. Essa regra identifica regras SWRL que possuem uma redundância nos seus átomos do antecedente. Essa redundância é causada pelo uso de uma classe e uma de suas subclasses em dois átomos do consequente da mesma regra.

Código-fonte 3 – Identificação da redundância *Redundant derivation in the antecedent* com o Datalog*

```
1: anomaly(redundant_derivation, Bs=>A) :-
2:   rule(Bs=>A), sub_sequence([Bi, Bj], Bs),
3:   (implies(Bi, Bj) ; implies(Bj, Bi)).
```

Fonte: Baumeister e Seipel (2010).

Para provar a capacidade dessa linguagem, os autores apresentaram as regras escritas em Datalog* para identificar 3 tipos de circularidades, 5 tipos de inconsistências, 9 tipos de redundâncias e 5 tipos de deficiências. Além disso, é possível elaborar novas regras de identificação no Datalog*.

As principais limitações encontradas nesse trabalho são:

- O uso de uma linguagem não comumente usada por desenvolvedores de ontologias para extrair informações de ontologias. Como uma anomalia nada mais é do que uma informação, com problemas, encontrada em ontologias, poderia ser usada uma linguagem já utilizada por desenvolvedores de ontologias para realizar essa identificação;
- A falta de suporte à análises em built-ins, como destacado pelos próprios autores;
- A falta de experimentos com um conjunto variado (representativo) de ontologias, como as encontradas em repositórios públicos na Web, para comprovar a necessidade de ferramentas para este fim.

3.2.2 *Ontology Evaluation: a pitfall-based approach to ontology diagnosis*

Este trabalho relacionado é uma sequência de publicações (POVEDA-VILLALÓN; SUÁREZ-FIGUEROA; GÓMEZ-PÉREZ, 2012), (KEET; SUÁREZ-FIGUEROA; POVEDA-VILLALÓN, 2013), (SUAREZ-FIGUEROA; KAMEL; POVEDA-VILLALON, 2013) e (POVEDA-VILLALÓN; GÓMEZ-PÉREZ; SUÁREZ-FIGUEROA, 2014) que culminaram em uma tese de doutorado (POVEDA-VILLALÓN, 2016).

Nesses trabalhos, o objetivo foi ajudar os desenvolvedores de ontologias a realizar diagnósticos em ontologias para diminuir o esforço requerido na validação dessas ontologias. As principais contribuições do trabalho foram a criação de um catálogo de anomalias (ou *Pitfalls*), que podem ocorrer no desenvolvimento de ontologias, e a implementação de métodos de detecção de anomalias para o diagnóstico (semi)automático de ontologias (POVEDA-VILLALÓN, 2016). O trabalho destaca que o catálogo elaborado não pretendeu ser extenso e que poderão ser adicionadas novas anomalias no futuro. No total, foram incluídas 41 anomalias ao catálogo, desde anomalias que são consideradas apenas recomendações até situações mais complexas que geram inconsistências.

As avaliações realizadas no trabalho se baseiam em avaliações qualitativas, sendo que as avaliações quantitativas estão previstas para o futuro (POVEDA-VILLALÓN, 2016). Para validação do trabalho, o método de detecção foi disponibilizado em uma ferramenta online para usuários de ontologias. Esses usuários puderam rodar a detecção em suas ontologias e validar a ferramenta respondendo a um questionário. Além disso, a ferramenta foi comparada a outras ferramentas nos quesitos funcionalidades e anomalias disponíveis para identificação.

As principais limitações desses trabalhos são:

- O número limitado de anomalias verificadas e a impossibilidade de extensão por desenvolvedores de ontologias;
- A não realização de identificações em anomalias que ocorrem em regras SWRL;
- A falta de experimentos controlados com um conjunto representativo de ontologias. Nos experimentos realizados com ontologias, os usuários submeterem suas ontologias para os experimentos, então não é possível saber a origem e a representatividade das ontologias analisadas. Além disso, não é possível garantir que não existem ontologias anônimas (que não tem a URI definida (POVEDA-VILLALÓN, 2016)) repetidas, com pequenas alterações, nos experimentos. Por isso, a autora destaca que são análises qualitativas da ferramenta desenvolvida (POVEDA-VILLALÓN, 2016).

3.2.3 *Discovering Anomalies in Semantic Web Rules*

Em *Discovering Anomalies in Semantic Web Rules*, [Feng et al. \(2010\)](#) propuseram um método para descobrir anomalias em regras SWRL de ontologias OWL DL usando uma técnica de *Constraint Logic Programming* (CLP). O método foi implementado em uma ferramenta, apresentada em trabalho prévio de [Dong et al. \(2005\)](#). Essa implementação identifica 2 tipos de circularidades, 5 tipos de inconsistências e 3 tipos de redundâncias.

Para validar o trabalho, os autores apresentam um estudo de caso em que aplicam o método desenvolvido na ontologia chamada *Context Ontology* (CONON) ([WANG et al., 2004](#)) e encontram ocorrências em 4 tipos de anomalias.

As principais limitações desse trabalho são:

- O número limitado de anomalias verificadas e a impossibilidade de extensão por desenvolvedores de ontologias;
- A falta de experimentos com várias ontologias representativas;
- Não realizar a identificação em anomalias que ocorrem em outras estruturas da ontologia além de suas regras, como classes, propriedades, etc.

3.2.4 *A Method of Eliminating Direct Super-Class to Detect Cycle of Concepts*

[Li et al. \(2009\)](#) propuseram um método para detectar ciclos de conceitos em ontologias que podem ser anomalias. O método se utiliza da montagem de um grafo dos conceitos da ontologia e, nesse grafo, o método procura por ocorrências de ciclos que formem as anomalias. No total, o trabalho apresenta um resumo de 19 anomalias apresentadas em outro trabalho ([GÓMEZ-PÉREZ, 1999](#)).

Para validar o trabalho, os autores apresentaram o funcionamento do método em uma ontologia de conceitos relacionados com relações familiares. O método identificou os ciclos de conceitos, porém ele não conseguiu julgar as características dos erros de forma concreta, como afirmado pelo autores ([LI et al., 2009](#)).

As principais limitações desse trabalho são:

- O número limitado de anomalias verificadas e a impossibilidade de extensão por desenvolvedores de ontologias;
- A verificação apenas de anomalias que geram ciclos nos conceitos;
- A verificação de anomalias extremamente básicas, que a própria especificação mais recente do OWL 2 resolve por si só. Por exemplo, a anomalia *Redundancies of*

subclass-of relations que ocorre quando existe duplicidade na definição de uma mesma relação de `rdfs:subClassOf`. O OWL 2, na sua especificação estrutural, define que todas as associações são conjuntos de objetos não ordenados e não repetidos (MOTIK *et al.*, 2009);

- A falta de experimentos com um conjunto representativo de ontologias;
- Não realizar a identificação de anomalias que ocorrem em regras SWRL.

3.2.5 *Managing and Refining Rule Sets for SWRL*

Sun *et al.* (2011) apresentaram um algoritmo para identificar e eliminar ocorrências de quatro tipos de redundâncias que ocorrem em regras SWRL. O algoritmo consegue detectar redundâncias que são geradas na combinação de várias regras. Esse algoritmo foi implementado em uma ferramenta e validado com a identificação de redundâncias em uma ontologia de teste. Além disso, nesse trabalho foram propostos outros algoritmos para identificar outros tipos de anomalias que também ocorrem em regras SWRL.

As principais limitações desse trabalho são:

- O número limitado de anomalias verificadas e a impossibilidade de extensão por desenvolvedores de ontologias;
- A somente verificação de anomalias que geram redundâncias;
- A verificação de redundâncias extremamente básicas, como duas regras que tem os mesmos átomos, mas em ordens diferentes;
- A falta de experimentos com ontologias representativas;
- A não realização de identificação em anomalias que ocorrem em outras estruturas da ontologia, somente nas regras.

3.2.6 *Elimination of Redundancy in Ontologies*

Grimm e Wissmann (2011) apresentam dois métodos para eliminar redundâncias em ontologias. Esses métodos usam uma técnica que mostra explicações detalhadas sobre as implicações em ontologias, chamada *Justifications*. Essa técnica mostra um conjunto mínimo de axiomas de uma ontologia que suporta uma determinada implicação realizada. Com isso, eles determinam quais informações não são necessárias e podem ser eliminadas em uma ontologia.

Para verificação desse método, os autores eliminaram as redundâncias em 12 importantes ontologias e reportaram a eficiência do algoritmo e os efeitos dessas eliminações no desempenho da ontologia.

As principais limitações desse trabalho são:

- Somente verifica anomalias que geram redundâncias;
- Não indica quais redundâncias poderão ser eliminadas. Segundo Grimm e Wissmann (2011), podem existir cenários em que uma determinada redundância pode ser uma característica desejável;
- Não realiza verificações em regras SWRL.

3.2.7 *OntoCheck: verifying ontology naming conventions and meta-data completeness in Protégé 4*

OntoCheck é um plug-in para o editor de ontologias Protégé que ajuda a limpar ontologias em relação às suas heterogeneidades lexicais. Isto é feito através da aplicação de convenções de nomenclaturas e integridade nos metadados (SCHOBER *et al.*, 2012).

Para a verificação da integridade de metadados, o sistema fornece um painel de configuração que permite aos usuários definirem as propriedades de anotações que devem ser verificadas. Também permite configurar quais convenções de nomenclatura ortográfica são usadas. As convenções de nomeação incluem separadores de palavras, padrão de uso de maiúsculo ou minúsculo, mínimo ou máximo de caracteres e contagem de palavras.

O plug-in foi testado com seis ontologias importantes e foi capaz de identificar diversas ocorrências de falta de padronização nas nomenclaturas.

As principais limitações desse trabalho são:

- O número limitado de verificações, apesar de permitir escolher quais serão identificadas;
- Somente verifica anomalias que ocorrem em convenções de nomes e em metadados, não suportando outras anomalias;
- A descontinuidade do plug-in. Ele só está disponível até a versão 4.1 do Protégé.

3.2.8 *XD Analyzer*

O XD Analyzer faz parte do plug-in XD Tools⁴ para o editor de ontologias NeOn Toolkit⁵ (DAGA *et al.*, 2010). O XD Analyzer fornece suporte de boas práticas em design de ontologias para desenvolvedores. Ele provê verificações em três diferentes classificações: “*error*”, “*warning*” e “*suggestion*”, respectivamente erro, aviso e sugestão.

⁴ XD Tools: <http://neon-toolkit.org/wiki/XDTools>

⁵ NeOn Toolkit: http://neon-toolkit.org/wiki/Main_Page

O XD Analyzer verifica 2 tipos de erros, 5 tipos de avisos e 2 tipos de sugestões em ontologias OWL (DAGA *et al.*, 2010).

As principais limitações desse trabalho são:

- O número limitado de anomalias verificadas e a impossibilidade de extensão por desenvolvedores de ontologias;
- As verificações são restritas apenas anomalias que são boas práticas em design de ontologias.

3.2.9 *Evaluation of Taxonomic Knowledge in Ontologies and Knowledge Bases*

Gómez-Pérez (1999) realizou um dos esforços iniciais para estabelecer uma lista de anomalias que podem ocorrer na construção de ontologias. Posteriormente, outros trabalhos surgiram com esse mesmo objetivo: (NOY; MCGUINNESS, 2001), (GÓMEZ-PÉREZ, 2001), (RECTOR *et al.*, 2004), (FAHAD; QADIR, 2008), (PAK; ZHOU, 2009), (ROUSSEY; CORCHO; VILCHES-BLÁZQUEZ, 2009), (CORCHO *et al.*, 2009) (VILLALON; SUÁREZ-FIGUEROA; GÓMEZ-PÉREZ, 2010) e (HOGAN *et al.*, 2010).

Apesar das importantes contribuições para o estabelecimento de uma lista de anomalias que devem ser evitadas em ontologias, e assim tornar as ontologias melhores, a principal limitação dessas abordagens é que as verificações devem ser realizadas de forma manual pelos desenvolvedores das ontologias.

3.3 **Resumo das limitações dos trabalhos relacionados**

A lista de limitações encontradas nos trabalhos relacionados foi a seguinte:

1. Trabalhos que não identificam anomalias em ontologias OWL;
2. Trabalhos que não identificam anomalias em regras SWRL;
3. Trabalhos que não permitem que o conjunto de anomalias identificadas possa ser estendido por desenvolvedores de ontologias;
4. Trabalhos que não permitem que os desenvolvedores estendam as anomalias identificadas usando linguagens bem conhecidas pela comunidade da Web Semântica;
5. Trabalhos que não permitem visualizar graficamente as anomalias identificadas;
6. Trabalhos que não realizam análises quantitativas da real ocorrência de anomalias em conjuntos representativos de ontologias.

A Tabela 2 mostra a relação entre os trabalhos relacionados e as limitações listadas acima. As limitações são numeradas de acordo com sua ordem na lista. As linhas da tabela representam os trabalhos relacionados identificados pela sua citação principal. A ordem dos trabalhos relacionados é a mesma que foi apresentada na subseção anterior.

Tabela 2 – Limitações encontradas nos trabalhos relacionados

Trabalho/Limitação	1	2	3	4	5	6
Baumeister e Seipel (2010)				X	X	X
Poveda-Villalón (2016)		X	X	X	X	Parcial
Feng <i>et al.</i> (2010)	X		X	X	X	X
Li <i>et al.</i> (2009)		X	X	X	X	X
Sun <i>et al.</i> (2011)	X		X	X	X	X
Grimm e Wissmann (2011)		X	X	X	X	Parcial
Schober <i>et al.</i> (2012)		X	X	X	X	Parcial
Daga <i>et al.</i> (2010)		X	X	X	X	X
Gómez-Pérez (1999)		X	X	X	X	X

Fonte: Elaborada pelo autor.

Na Tabela 2 pode ser observado que a limitação 1 (identificação de anomalias em ontologias OWL) ocorre apenas em dois trabalhos que só implementam as identificações para regras SWRL, sem identificar as partes de uma ontologia OWL. O único trabalho que não sofreu as limitações 1 e 2 em conjunto foi o de Baumeister e Seipel (2010). Esse mesmo trabalho é o único que permite a implementação da identificação de novas anomalias. O limite desse trabalho, como todos os demais, é a impossibilidade de estender as identificações usando uma linguagem amplamente utilizada na comunidade da Web Semântica (como OWL, SPARQL ou SWRL). Outro ponto importante dessa comparação é que nenhum dos trabalhos relacionados possui visualização gráfica das anomalias encontradas.

Por último, quanto à limitação 6, os trabalhos relacionados de Poveda-Villalón (2016), Grimm e Wissmann (2011) e Schober *et al.* (2012) são os únicos que possuem, parcialmente, verificações controladas em conjuntos representativos de ontologias. Em Poveda-Villalón (2016), os experimentos em ontologias foram realizados por usuários que submeteram ontologias para o sistema, não garantindo a replicabilidade dos resultados. Em Schober *et al.* (2012) e Grimm e Wissmann (2011), apesar dos experimentos serem controlados e replicáveis, foram realizados, respectivamente, em apenas 6 e 12 ontologias escolhidas pelos autores dos trabalhos.

3.4 Conjunto de anomalias escolhidas da literatura científica

A partir dos trabalhos relacionados foram selecionadas anomalias que servirão para demonstrar o método de identificação de anomalias proposto neste trabalho. Essas anomalias são um conjunto representativo de anomalias que ocorrem em ontologias. Para ser um conjunto representativo, as anomalias escolhidas foram distribuídas entre diferentes classificações, diferentes entidades afetadas e diferentes graus de complexidades.

Então para selecionar as anomalias foram estabelecidos os 3 seguintes critérios:

- As anomalias escolhidas deverão ocorrer nos diversos tipos de entidades de uma ontologia, como classes, propriedades, cardinalidades e regras SWRL, para provar que o método proposto é capaz de identificar ocorrências em diferentes tipos de entidades.
- As anomalias escolhidas deverão pertencer às 4 classificações propostas por [Baumeister e Seipel \(2010\)](#) para provar que o método proposto é capaz de identificar diferentes tipos de anomalias (Tipos de anomalias foram apresentados na Seção 3.1).
- O conjunto de anomalias escolhidas deverá conter anomalias simples e complexas, para provar que o método proposto é capaz de identificar anomalias de ambos os tipos.

Seguindo esses critérios, uma lista com dezoito anomalias foi escolhida e é apresentada nesta seção. De forma resumida, a Tabela 3 lista as anomalias escolhidas, destacando, para cada anomalia, a principal entidade envolvida e a classificação segundo [Baumeister e Seipel \(2010\)](#). Em relação à entidade envolvida na anomalia, foram escolhidas 6 anomalias que ocorrem em classes, 4 em propriedades, 2 em cardinalidades e 9 em regras (3 anomalias ocorrem tanto em classes como propriedades). Em relação a classificação da anomalia, foram escolhidas 3 circularidades, 7 redundâncias, 5 inconsistências e 3 deficiências.

Nas próximas subseções, cada uma das anomalias da Tabela 3 será explicada mais detalhadamente.

3.4.1 *Chains of inheritance*

As hierarquias de classes definem a espinha dorsal de cada ontologia. As relações de subclasse são usadas para descrever a herança entre as classes. Durante a criação da hierarquia de classes ou durante a integração de várias ontologias, podem ser criadas cascatas de subclasses ([BAUMEISTER; SEIPEL, 2010](#); [FAHAD; QADIR, 2008](#)). Se essas

Tabela 3 – Lista das anomalias escolhidas. Incluindo nome, entidade(s) afetada(s) e tipo.

Nome da Anomalia	Principal entidade envolvida	Tipo da Anomalia
<i>Chains of inheritance</i>	Classes	Deficiência
<i>Lonely disjoint classes</i>	Classes	Deficiência
<i>Partition error in taxonomy</i>	Classes	Inconsistência
<i>Exact circularity in taxonomy</i>	Classes ou propriedades	Circularidade
<i>Redundancy by repetitive taxonomic definition</i>	Classes ou propriedades	Redundância
<i>Lazy class or property</i>	Classes ou propriedades	Deficiência
<i>Circular properties</i>	Propriedades de objetos	Circularidade
<i>Multiple functional properties</i>	Cardinalidades	Inconsistência
<i>Redundant cardinalities</i>	Cardinalidades	Redundância
<i>Circularity between rules and taxonomy</i>	Regras	Circularidade
<i>Contradicting rules</i>	Regras	Inconsistência
<i>Incompatible rule antecedent</i>	Regras	Inconsistência
<i>Redundant derivation in the antecedent</i>	Regras	Redundância
<i>Redundant implication</i>	Regras	Redundância
<i>Redundant implication of transitivity or symmetry</i>	Regras	Redundância
<i>Redundant use of transitivity and symmetry</i>	Regras	Redundância
<i>Self-contradicting rule</i>	Regras	Inconsistência
<i>Rule subsumption</i>	Regras	Redundância

Fonte: Elaborada pelo autor.

cascatas de subclasses não possuem utilidade em nenhuma outra parte da ontologia, isso pode configurar a ocorrência dessa anomalia. Para classificar uma cascata de subclasses como uma *Chains of inheritance*, é necessário atender as seguintes condições, segundo Baumeister e Seipel (2010):

- Devem ser três ou mais classes em cascata (Classe A tem subclasse B que tem subclasse C);
- Não devem existir instâncias nessas classes ou terem poucas instâncias;
- Não devem existir restrições ou outras definições usando essas classes;
- Não devem existir regras usando essas classes.

3.4.2 *Lonely disjoint classes*

Esta anomalia ocorre quando uma classe não é disjunta de nenhuma das suas classes irmãs (classes com a mesma super classe) e é disjunta de pelo menos duas classes de outro ramo da hierarquia de classes (BAUMEISTER; SEIPEL, 2010; FAHAD; QADIR,

2008). Na Figura 4 são mostrados os dois ramos de classes A e B de uma ontologia, em que a classe A1 é disjunta das classes B2 e B3 que são encontradas em outro ramo da hierarquia.

Figura 4 – Exemplo de uma ocorrência da anomalia *Lonely disjoint classes*

Fonte: Adaptada de Baumeister e Seipel (2010).

Uma *Lonely disjoint classes* é muitas vezes criada pela modificação manual da ontologia quando uma classe é movida para outro ramo da taxonomia, mas as definições de disjunção não são realinhadas adequadamente (BAUMEISTER; SEIPEL, 2010). Além disso, essa anomalia também pode ocorrer devido a alinhamentos incorretos durante a integração de várias ontologias (BAUMEISTER; SEIPEL, 2010). A existência dessa anomalia pode causar resultados de raciocínio não intencional ou mesmo erros na ontologia (BAUMEISTER; SEIPEL, 2010).

3.4.3 Partition error in taxonomy

O *Partition error in taxonomy* ocorre quando existe a combinação incorreta de disjunção e derivação (herança) (GÓMEZ-PÉREZ, 1999; BAUMEISTER; SEIPEL, 2010; FAHAD; QADIR, 2008). Essa anomalia é encontrada quando duas classes disjuntas possuem uma mesma subclasse. Dado que toda instância que pertence a uma classe X também pertence às superclasses de X, a classe X nunca poderá ter instâncias quando às superclasses forem disjuntas.

Na Figura 5 é apresentada uma ocorrência dessa anomalia, que é descrita por Gómez-Pérez (1999). Nessa figura, a classe *Prime* possui as superclasses *Odd* e *Even* que são disjuntas.

Figura 5 – Exemplo da anomalia *Partition error in taxonomy*

Fonte: Elaborada pelo autor.

3.4.4 Exact circularity in taxonomy

Esta anomalia ocorre quando classes ou propriedades são alinhadas de forma incorreta (BAUMEISTER; SEIPEL, 2010). Para isso ocorrer, devem existir duas entidades (classes ou propriedades) equivalentes, C1 e C2, onde a entidade C1 é superentidade de S1, a entidade C2 é subentidade de S2 e as entidades S1 e S2 são equivalentes. Como pode ser visto na Figura 6, esses alinhamentos formam essa circularidade. Essa é a forma mais simples dessa circularidade, porém podem ocorrer situações em que existem várias entidades intermediárias que tornam muito difícil a descoberta dessas ocorrências. A consequência dessa anomalia é que todas as entidades envolvidas, até mesmo as entidades intermediárias, serão consideradas como equivalentes.

Figura 6 – Exemplo de uma ocorrência da anomalia *Exact circularity in taxonomy*

Fonte: Elaborada pelo autor.

3.4.5 Redundancy by repetitive taxonomic definition

Esta anomalia é a definição redundante de `rdfs:subClassOf` ou `rdfs:subPropertyOf` em hierarquias de classes ou propriedades (GÓMEZ-PÉREZ, 1999). Essa anomalia pode ocorrer de duas forma (GÓMEZ-PÉREZ, 1999):

- Repetição direta: Ocorre quando duas ou mais definições de `rdfs:subClassOf` ou `rdfs:subPropertyOf` são definidas entre as mesmas classes ou propriedades.
- Repetição indireta: Ocorre quando um conjunto de definições `rdfs:subClassOf` ou `rdfs:subPropertyOf` apresenta a redundância na hierarquia em que uma das definições pode ser deduzida pelo resto das definições. Para ilustrar essa variação da anomalia, a Figura 7 apresenta 3 classes e 3 definições de `rdfs:subClassOf` em que a definição de **Father** como `rdfs:subClassOf` de **Person** é a redundância. Isso é uma redundância, pois, a partir da definição de que **Parent** é uma `rdfs:subClassOf` de **Person**, é possível inferir que todas subclasses de **Parent** também são subclasses de **Person**.

Figura 7 – Exemplo de uma ocorrência da anomalia *Redundancy by repetitive taxonomic definition*

Fonte: Elaborada pelo autor.

3.4.6 Lazy class or property

Segundo [Baumeister e Seipel \(2010\)](#), uma classe ou propriedade é considerada “preguiçosa” quando nunca ou raramente é usada em aplicações reais. Ainda segundo os mesmos autores, os critérios para definir essa anomalia são os seguintes:

- A entidade é um folha na hierarquia;
- A entidade não é usada por nenhuma regra;
- A entidade não tem instâncias;
- A entidade não é usada em restrições de outras entidades.

3.4.7 Circular properties

Esta anomalia ocorre quando existem duas propriedades inversas que têm o mesmo ou equivalente `rdfs:domain` e o mesmo ou equivalente `rdfs:range` (BAUMEISTER; SEIPEL, 2010). Por exemplo, na Figura 8 (exemplo descrito por Baumeister e Seipel (2010)) existem duas ontologias A e B que possuem as seguintes definições de alinhamento de classes: `A:Lecture` é equivalente à `B:Course` e `A:Professor` é equivalente à `B:Professor`. Além disso, nessas mesmas ontologias, existem as seguintes propriedades: A propriedade `A:lectures` com o `rdfs:domain` igual a `A:Professor` e o `rdfs:range` igual a `A:Lecture`; e a propriedade `B:teaches` com o `rdfs:domain` igual a `B:Professor` e o `rdfs:range` igual a `B:Course`. Uma *circular property* será definida, nesse caso, se existir uma definição que `A:lectures` é inversa à `B:teaches`.

Figura 8 – Exemplo de uma ocorrência da anomalia *Circular properties*

Fonte: Elaborada pelo autor.

3.4.8 Multiple functional properties

Uma propriedade definida como funcional só pode ter uma ocorrência por instância. Portanto, a definição de uma propriedade funcional pode ser canonicamente traduzida para uma propriedade com uma restrição de cardinalidade mínima maior ou igual a 0 e uma restrição de cardinalidade máxima menor ou igual a 1. Porém, desenvolvedores de ontologias podem definir um máximo maior que 1 para essa restrição (BAUMEISTER; SEIPEL, 2010). Nesse caso, essa definição gera uma contradição na ontologia.

3.4.9 Redundant cardinalities

As definições de cardinalidades podem ser redundantes em algumas situações. Baumeister e Seipel (2010) citam, em especial, duas situações em que isso pode ocorrer:

- Ao definir uma propriedade com a restrição de cardinalidade mínima maior ou igual a zero;
- Ao definir uma propriedade com a restrição de cardinalidade máxima igual a 1 e que possua uma super propriedade definida como funcional.

3.4.10 *Circularity between rules and taxonomy*

Essa anomalia, do tipo circularidade, ocorre em regras SWRL quando um átomo do conseqüente implica um átomo do antecedente (BAUMEISTER; SEIPEL, 2010). A implicação, nesse caso, afirma que toda instância classificada no átomo do conseqüente também é classificada no átomo do antecedente. Isso ocorre com o uso de uma classe no antecedente da regra e sua subclasse no conseqüente da regra, gerando assim uma circularidade. Para demonstrar uma ocorrência dessa anomalia, o código-fonte 4 apresenta uma regra que tem essa circularidade caso **Father** seja definido como uma subclasse de **Person**. A regra deveria ser considerada como uma relação `rdfs:subClassOf` restrita entre **Father** e **Person** sem gerar a circularidade. A definição de uma relação de `rdfs:subClassOf` usando uma regra pode ser uma definição taxonômica mal aplicada entre essas classes (BAUMEISTER; SEIPEL, 2010).

Código-fonte 4 – Exemplo de uma ocorrência da anomalia *Circularity between rules and taxonomy*

```
1: Person(?x) ^ hasChild(?x, ?y) -> Father(?x)
```

Fonte: Elaborada pelo autor.

3.4.11 *Contradicting rules*

Regras contraditórias ocorrem quando duas regras SWRL têm os átomos do antecedente iguais ou equivalentes e dois átomos do conseqüente (um de cada regra) que possuem predicados disjuntos (BAUMEISTER; SEIPEL, 2010; FENG *et al.*, 2010). Se as regras forem executadas, uma contradição será gerada, porém, para essas regras gerarem um resultado contraditório, é necessário que existam instâncias que cumpram as condições dos antecedentes das regras. Um *reasoner* detectaria essa contradição somente se as regras gerarem resultados. Ou seja, podem existir regras contraditórias, que não geram nenhum resultado, em ontologias que foram verificadas por um *reasoner*.

Para ilustrar a ocorrência dessa anomalia, o Código-fonte 5 apresenta duas regras SWRL que podem apresentar essa anomalia. Além da definição das regras, a ontologia deverá conter os seguintes alinhamentos:

- A classe **a:Person** é equivalente à classe **b:Person**;

- A classe `a:Lecture` é equivalente à classe `b:Lecture`;
- A propriedade `a:lectures` é equivalente à propriedade `b:inLectures`;
- A classe `a:Teacher` é disjunta da `b:Student`.

A definição incorreta de equivalência entre as propriedades `a:lectures` e `b:inLectures` gera o conhecimento contraditório.

Código-fonte 5 – Exemplo de uma ocorrência da anomalia *Contradicting rules*

```
1: a:Person(X) ^ a:Lecture(Y) ^ a:lectures(X,Y) -> a:Teacher(X)
2: b:Person(X) ^ b:Lecture(Y) ^ b:inLecture(X,Y) -> b:Student(X),
```

Fonte: Adaptada de [Baumeister e Seipel \(2010\)](#).

3.4.12 Redundant derivation in the antecedent

Essa anomalia ocorre em uma regra SWRL quando um átomo do antecedente implica em outro átomo do antecedente da mesma regra ([BAUMEISTER; SEIPEL, 2010](#); [FENG et al., 2010](#)). A regra do Código-fonte 6 conterá essa redundância se existir a definição de que `Father` é subclasse de `Person`. Nesse caso, o átomo `Person(?x)` é redundante e não necessário para obter o mesmo resultado da regra, pois toda instância pertencente a classe `Father` sempre será pertencente a sua super classe `Person`.

Código-fonte 6 – Exemplo de uma ocorrência da anomalia *Redundant derivation in the antecedent*

```
1: Person(?x) ^ Father(?x) ^ hasChild(?x, ?y) -> Child(?y)
```

Fonte: Elaborada pelo autor.

3.4.13 Redundant implication

Essa anomalia acontece quando um átomo do antecedente implica em um átomo do consequente ([BAUMEISTER; SEIPEL, 2010](#)). Essa redundância é similar a *Circularity between rules and taxonomy* (Subseção 3.4.10), mas, nesse caso, a relação de subclasse é invertida: um átomo do antecedente implica em um átomo do consequente. O átomo do consequente não é necessário, pois a definição da taxonomia já garante essa classificação. No Código-fonte 7, levando em consideração que exista a definição de que `Person` é superclasse de `Father`, toda instância `X` que pertence a classe `Father` também pertence a sua superclasse `Person`, então essa regra executa algo que já é deduzido por meio da taxonomia da ontologia.

Código-fonte 7 – Exemplo de uma ocorrência da anomalia *Redundant implication*

```
1: Father(?x) ^ hasChild(?x, ?y) -> Person(?x)
```

Fonte: Elaborada pelo autor.

3.4.14 *Redundant implication of transitivity or symmetry*

Essa anomalia, que ocorre em regras SWRL, pode ser dividida em duas situações, segundo [Baumeister e Seipel \(2010\)](#):

Propriedades Transitivas: Quando uma propriedade transitiva possui os arranjos $P(x, y)$ e $P(y, z)$, *reasoners* podem inferir $P(x, z)$. A anomalia *Redundant Implication of Transitivity* acontece quando uma regra tem pelo menos 3 propriedades P1, P2 e P3 que são equivalentes e transitivas e usa P1 e P2 para afirmar P3: $P1(?x, ?y) \wedge P2(?y, ?z) \wedge \dots \rightarrow P3(?x, ?z)$. Apesar de correto, a regra ou o átomo do consequente não é necessário na ontologia.

Propriedades Simétricas: Quando uma propriedade simétrica possui o arranjo $P(x, y)$, *reasoners* podem inferir $P(y, x)$. A anomalia *Redundant Implication of Symmetry* acontece quando uma regra tem pelo menos 2 propriedades P1 e P2, que são simétricas, e usa P1 para afirmar P2: $P1(?x, ?y) \wedge \dots \rightarrow P2(?y, ?x)$. Apesar de correto, a regra ou o átomo do consequente não é necessário na ontologia.

3.4.15 *Redundant use of transitivity and symmetry*

Essa anomalia é similar a anterior (Subseção 3.4.14) e também pode ser dividida em duas situações, segundo [Baumeister e Seipel \(2010\)](#):

Propriedades Transitivas: A anomalia *Redundant use of transitivity* ocorre quando uma regra tem pelo menos 3 propriedades P1, P2 e P3 que são equivalentes e transitivas, e usa P1, P2 e P3 no antecedente: $P1(?x, ?y) \wedge P2(?y, ?z) \wedge P3(?x, ?z) \wedge \dots \rightarrow \dots$. O átomo $P3(?x, ?z)$ não é necessário na regra.

Propriedades Simétricas: A anomalia *Redundant use of symmetry* ocorre quando uma regra tem pelo menos 2 propriedades P1 e P2, que são simétricas, e usa P1 e P2 no antecedente: $P1(?x, ?y) \wedge P2(?y, ?x) \wedge \dots \rightarrow \dots$. O átomo $P1(?x, ?y)$ ou o átomo $P2(?y, ?x)$ pode ser retirado da regra.

3.4.16 *Incompatible rule antecedent*

Essa anomalia, que ocorre em regras SWRL, acontece quando existe uma incompatibilidade entre os átomos do antecedente de uma regra ([BAUMEISTER; SEIPEL, 2010](#);

FENG *et al.*, 2010). A incompatibilidade é gerada a partir de dois átomos do antecedente de uma regra, os quais possuem dois predicados disjuntos e usam as mesmas variáveis.

Para exemplificar, o Código-fonte 8 apresenta partes de duas regras que contêm essa anomalia. Na primeira regra, os átomos `Father(?x)` e `Mother(?x)` possuem, como predicados, duas classes disjuntas `Father` e `Mother`, e buscam por instâncias que pertençam a essas classes disjuntas. A consequência dessa regra é que ela nunca será executada, pois para isso acontecer todas as afirmações do antecedente deveriam ser verdadeiras e, nesse caso, uma instância não pode pertencer ao mesmo tempo às classes `Father` e `Mother`. Na segunda regra acontece algo similar à primeira, porém a diferença é que são propriedades disjuntas que afirmam que um indivíduo X tem um pai Y e um filho Y.

A consequência dessa anomalia é que nunca o antecedente será todo verdadeiro, ou seja, a regra nunca irá gerar resultados.

Código-fonte 8 – Exemplos de duas ocorrências da anomalia *Incompatible rule antecedent*

```
1: Father(?x) ^ Mother(?x) ^ ... -> ...
2: hasParent(?x, ?y) ^ hasChild(?x, ?y) ^ ... -> ...
```

Fonte: Elaborada pelo autor.

3.4.17 Self-contradicting rule

Essa anomalia é similar a anomalia *Incompatible rule antecedent* (Seção 3.4.16), com a diferença que a incompatibilidade de átomos ocorre entre um átomo do consequente e outro do antecedente. Diferentemente da anterior, essa anomalia irá gerar uma inconsistência no resultado da execução da regra. Para exemplificar essa anomalia, o Código-fonte 9 apresenta duas regras similares as da seção anterior. Ambas tentam realizar conclusões inconsistentes na ontologia e devem ser corrigidas.

Código-fonte 9 – Exemplos de duas ocorrências da anomalia *Self-contradicting rule*

```
1: Father(?x) ^ ... -> Mother(?x)
2: hasParent(?x, ?y) ^ ... -> hasChild(?x, ?y)
```

Fonte: Elaborada pelo autor.

3.4.18 Rule subsumption

Essa anomalia ocorre quando o antecedente de uma regra X pode ser totalmente mapeado no antecedente de uma outra regra Y e os consequentes dessas regras são iguais

ou equivalentes (BAUMEISTER; SEIPEL, 2010; SUN *et al.*, 2011; FENG *et al.*, 2010). Como a regra X possui uma condição menos restritiva que a regra Y, todas as instâncias que atendam o antecedente da regra Y também vão atender o antecedente da regra X. Como essas regras possuem o mesmo consequente, a regra Y é uma redundância na ontologia e pode ser removida.

Para exemplificar essa anomalia, o Código-fonte 10 apresenta duas regras em que essa anomalia ocorre. Na primeira regra todas as pessoas que tem 21 anos ou mais são classificadas como adultos e na segunda as pessoas com 18 anos ou mais também são assim classificadas. Como pode ser observado, a segunda regra abrange mais instâncias que a primeira, tornando a primeira desnecessária.

Código-fonte 10 – Exemplos de duas ocorrências da anomalia *Rule subsumption*

```
1: Person(?x) ^ hasAge(?x, ?age), greaterThanOrEqualTo(?age, 21) ->
    Adult(?x)
2: Person(?x) ^ hasAge(?x, ?age), greaterThanOrEqualTo(?age, 18) ->
    Adult(?x)
```

Fonte: Elaborada pelo autor.

3.5 Considerações finais

O propósito deste capítulo foi apresentar a área de identificação de anomalias em ontologias e uma seleção de dezoito anomalias que serão detectadas pelo método proposto neste trabalho. Através da análise dos nove trabalhos relacionados (Seção 3.2), foram encontradas seis diferentes limitações. Apenas um trabalho supre três dessas limitações enquanto, os demais, só suprem totalmente uma das limitações. As limitações quatro⁶ e cinco⁷ foram encontradas em todos os trabalhos relacionados. Quanto à limitação seis⁸, nenhum trabalho realizou uma análise quantitativa sobre um conjunto representativo de ontologias e apenas três trabalhos realizaram essa análise de forma parcial.

Uma das limitações (Limitação 5), encontrada em todos os trabalhos relacionados, foi que as anomalias identificadas não são apresentadas em visualizações gráficas. Nenhum dos trabalhos relacionados focou na apresentação de anomalias. O próximo capítulo apresentará trabalhos relacionados com visualizações gráficas para ontologias.

⁶ Limitação 4: Trabalhos que não permitem que os desenvolvedores estendam as anomalias identificadas usando linguagens bem conhecidas pela comunidade da Web Semântica.

⁷ Limitação 5: Trabalhos que não permitem visualizar graficamente as anomalias identificadas.

⁸ Limitação 6: Trabalhos que não realizam análises quantitativa da real ocorrência de anomalias em grandes conjuntos de ontologias encontrados em repositórios online.

VISUALIZAÇÃO DA INFORMAÇÃO APLICADA À ONTOLOGIAS

Para corrigir as anomalias identificadas, desenvolvedores de ontologias devem entender como elas ocorreram e as estruturas afetadas. Contudo, anomalias podem estar ligadas a estruturas complexas e difíceis de entender. Além disso, atualmente ontologias são desenvolvidas por grupos de especialistas de várias áreas, incluindo usuários leigos em ontologias. Em especial, esses usuários leigos podem ter diversas dificuldades para entender uma anomalia encontrada em uma ontologia (LOHMANN *et al.*, 2014).

O uso de visualização da informação em ontologias pode ajudar nesse sentido, auxiliando na exploração, verificação e entendimento de determinadas partes de uma ontologia (GEROIMENKO; CHEN, 2006; LANZENBERGER; SAMPSON; RESTER, 2009). Visualizações podem ser particularmente úteis para usuários leigos, mas também podem dar a usuários especializados uma nova perspectiva sobre as ontologias (LOHMANN *et al.*, 2014).

De um modo geral, a visualização da informação é um campo de pesquisa bem estabelecido que tem por objetivo transformar informações em representações visuais (FU; NOY; STOREY, 2013). Essas representações visuais utilizam metáforas para facilitar a interpretação e compreensão de dados, a fim de fornecer ao usuário informações relevantes (DUDÁŠ; ZAMAZAL; SVÁTEK, 2014). A visualização tem um grande potencial quando se trata de criar, explorar ou verificar coleções complexas de dados e, com ontologias, isso não é diferente (LANZENBERGER; SAMPSON; RESTER, 2009).

Segundo Dudáš, Zamazal e Svátek (2014), uma visualização adequada pode ser essencial para o desenvolvimento, compartilhamento e utilização de uma ontologia. Ao longo dos anos, muitos métodos de visualização foram desenvolvidos para ontologias. No entanto, segundo Dudáš, Zamazal e Svátek (2014) nenhum método de visualização foi aceito pela

maioria dos usuários da web semântica como padrão. Ainda segundo os mesmos autores, uma razão para isso é que diferentes situações requerem diferentes métodos de visualização ontológica. Em sua grande maioria, os métodos de visualização para ontologias utilizam técnicas simples, como árvores e grafos bidimensionais, geralmente mostrando os conceitos ontológicos como nós e as relações ontológicas como arestas (LANZENBERGER; SAMPSON; RESTER, 2009).

Para apresentar as técnicas de visualização aplicadas a ontologias foram analisados os *surveys* de Dudáš, Zamazal e Svátek (2014), Lanzenberger, Sampson e Rester (2009) e Katifori *et al.* (2007), os trabalhos comparativos de Fu, Noy e Storey (2013), Guo e Chan (2011) e Katifori *et al.* (2006), e outros trabalhos mais recentes em visualização de ontologias.

Como se tratam de mais de 30 implementações diferentes de visualizações, foi decidido abordar somente os diferentes tipos de métodos de visualização aplicados em ontologias. Para tanto, será utilizada a classificação de tipos métodos aplicados em ontologias apresentada por Katifori *et al.* (2007):

1. *Node-link and tree*;
2. *Indented list*;
3. *Zoomable*;
4. *Space-filling*;
5. *Focus + context or distortion*;
6. *3D Information landscapes*.

Nas seguintes seções, serão apresentadas cada uma dessas classificações com pelo menos um método que a implementa e as suas vantagens/desvantagens.

4.1 Node-link and tree

Dados organizados em redes estruturadas, como ontologias, podem naturalmente ser representados em grafos ou árvores. De um modo geral, grafos são representações visuais que possuem nós que representam as instâncias dos dados e arestas que representam ligações entre os nós (MAZZA, 2009). Em ontologias, normalmente, os nós são as entidades de uma ontologia e as arestas são as ocorrências de determinadas relações entre as entidades. Em muitos casos, técnicas dessa categoria apresentam as ontologias como taxonomias em um layout de cima para baixo ou da esquerda para direita (KATIFORI *et al.*, 2007). O usuário, a partir dessas visualizações, pode expandir ou retrair árvores, a fim de ajustar o detalhamento da informação mostrada e evitar uma desordem na exibição.

Um exemplo, que usa esse tipo de técnica, é a visualização da ferramenta OntoGraf do Protégé, exibida na Figura 9. Nessa figura, as classes e as instâncias são representadas em nós na forma de retângulos e as diferentes ligações existentes entre essas entidades são representadas por linhas em diferentes estilos. Essa ferramenta pode ser usada somente para ontologias pequenas e as ligações entre as entidades, que são exibidas, são limitadas à expressividade do RDFS (DUDÁŠ; ZAMAZAL; SVÁTEK, 2014). Além disso, apesar do uso de várias cores para diferenciar essas ligações entre as entidades, ainda é necessário filtrar algumas ligações na ferramenta, de modo a se obter uma visualização mais clara.

Figura 9 – Exemplo de uma ontologia visualizada em forma de árvore no layout da esquerda para direita na ferramenta OntoGraf

Fonte: Elaborada pelo autor.

A principal vantagem deste método é que árvores e grafos são uma maneira comum e intuitiva de representação de hierarquias. Esse tipo de técnica permite ao usuário ter uma visão geral das estruturas hierárquicas, principalmente ao analisar itens como profundidade e largura (KATIFORI *et al.*, 2007). Por outro lado, esses métodos tipicamente fazem uso ineficiente do espaço de tela, deixando o lado da raiz completamente vazio e o lado oposto superlotado (PLAISANT; GROSJEAN; BEDERSON, 2002). Mesmo representações de uma centena de nós muitas vezes precisam de várias telas para serem exibidas completamente, ou exigem rolagem para casos em que o diagrama é visível parcialmente (KATIFORI *et al.*, 2007). Ham e Wijk (2002) e Bruls, Huizing e Wijk (2000) também

suportam a mesma afirmação de Katifori *et al.* (2007), ao afirmarem que os diagramas tradicionais de nós conduzem a visualizações desordenadas quando usadas para mais do que algumas centenas de nós.

Figura 10 – Exemplo de uma ontologia visualizada em forma de grafo direcionado pela força na ferramenta WebVOWL

Fonte: Elaborada pelo autor.

Para ter uma ocupação melhor do espaço, é possível visualizar a ontologia como um grafo direcionado pela força e usar técnicas de interação para explorar e customizar a visualização (LOHMANN *et al.*, 2014). Na Figura 10 é apresentado um exemplo dessa

representação, extraída da ferramenta WebVOWL (LOHMANN *et al.*, 2014). Nessa representação são usadas cores, símbolos, linhas e rótulos para representar todas as informações de uma ontologia. As principais desvantagens dessa representação são entidades repetidas (como *Thing*), representação não intuitiva de hierarquias e excesso de informação na visualização de ontologias complexas.

4.2 Indented list

Vários editores de ontologias, como o Protégé (NOY; FERGERSON; MUSEN, 2000) e o OntoEdit (SURE; ANGELE; STAAB, 2002), possuem pelo menos um painel para visualização da lista de entidades de uma ontologia. Normalmente nessas visualizações, as entidades das ontologias são representadas em listas indentadas. Na Figura 11 é apresentado, como exemplo, o navegador de classes do editor de ontologias Protégé. Nessa figura, as classes são apresentadas como nós em uma árvore indentada, expansível e retrátil. As classes filhas são colocadas sob a sua classe pai e recuadas para a direita.

Figura 11 – Exemplo de visualização de uma taxonomia em lista indentada

Fonte: Elaborada pelo autor.

As principais vantagens das visualizações dessa categoria são a sua simplicidade de implementação e representação, e a familiaridade dos usuários, pois ela é muito utilizada por navegadores de arquivos, como o *Windows Explorer* (KATIFORI *et al.*, 2007; KATIFORI *et al.*, 2006). Além disso, a simplicidade da interface torna essa visualização

conveniente para navegação rápida. Essa é provavelmente a razão pela qual ela tem sido tão eficaz na recuperação de informações e é a principal técnica utilizada para a edição de ontologias.

Uma limitação dessa técnica, é o fato dela usar representações em árvores e não em grafos. Como resultado disso, em ontologias ela exibe apenas relações de herança, e não relações de função. Além disso, os casos de herança múltipla não são muito óbvios. Por exemplo, o Protégé trata desses casos colocando o nó filho sob todos os seus pais. No entanto, nem sempre é claro, para um usuário inexperiente em ontologias, o real motivo de uma classe aparecer duas ou mais vezes em partes diferentes de uma hierarquia (KATIFORI *et al.*, 2007).

4.3 Zoomable

A categoria Zoomable contém todos os métodos que apresentam os nós filhos de uma hierarquia dentro do seus nós pais (KATIFORI *et al.*, 2007). As técnicas dessa categoria permitem que o usuário aplique zoom nos nós filhos, a fim de ampliá-los, tornando-os o nível de visualização atual. Um exemplo dessa técnica pode ser visto na ferramenta Jambalaya (Figura 12). A Jambalaya é um plug-in da ferramenta Protégé que representa uma estrutura hierárquica como um conjunto de classes que encapsulam outras classes ou instâncias (GUO; CHAN, 2011).

Esse tipo de visualização é eficaz para a localização de nós específicos, uma vez que proporciona uma visão mais abrangente do nível de hierarquia que o usuário estiver ampliando (KATIFORI *et al.*, 2007). Uma das limitações desse tipo de técnica é o uso de muitas ligações entre as classes ou instâncias na visualização, o que pode confundir o usuário (DUDÁŠ; ZAMAZAL; SVÁTEK, 2014).

4.4 Space-filling

Técnicas da categoria *Space-filling* baseiam-se no conceito de usar todo o espaço de tela e subdividir esse espaço disponível para todos os nós filhos de uma taxonomia (KATIFORI *et al.*, 2007). O tamanho de cada subdivisão corresponde a uma propriedade do nó, podendo ser seu tamanho, número de nós filhos, etc.

Um exemplo de visualização nessa categoria é o método de visualização chamado *TreeMaps* (SHNEIDERMAN, 1992). O *TreeMaps* usa uma abordagem de preenchimento de espaço para representar hierarquias, usando uma área retangular com subdivisões retangulares (Figura 13). Para ontologias, a técnica *Treemap* foi proposta por Baehrecke *et al.* (2004) como uma ferramenta para visualizar a ontologia GO (*Gene Ontology*). A técnica usou tamanhos e cores diferentes para fornecer um mecanismo de avaliação dos

Figura 12 – Exemplo de visualização da categoria Zoomable

Fonte: Elaborada pelo autor.

dados. O *Treemap* proposto tinha a funcionalidade de atribuir etiquetas, tamanho e cor a diferentes atributos genéticos.

Segundo [Plaisant, Grosjean e Bederson \(2002\)](#) e [Ham e Wijk \(2002\)](#), essa categoria de visualização é bem sucedida na visualização de árvores que possuam valores de propriedade no nível de nó folha (instância), como pode ser o caso de estruturas ontológicas. Principalmente, é muito eficaz quando o usuário tem maior interesse nos nós folhas e suas propriedades e não precisa se concentrar na topologia da árvore, ou quando a topologia é trivial, tendo no máximo 2 ou 3 níveis. Em relação às desvantagens, [Katifori et al. \(2007\)](#) apontam que, nos *Treemaps*, o principal problema é que não há espaço para os nós mais internos da árvore.

4.5 Focus + context or distortion

Este grupo de técnicas baseia-se na noção de distorcer a visualização apresentada para combinar contexto e foco ([KATIFORI et al., 2007](#)). O nó em foco é geralmente o

pode-se gerar uma desorientação no usuário.

4.6 3D Information landscapes

Uma metáfora muito comum usada em ambientes de realidade virtual para gerenciamento de documentos é a metáfora de paisagem. Nela, documentos são colocados em um plano como objetos 3D codificados por cores e tamanhos (KATIFORI *et al.*, 2007). Segundo Katifori *et al.* (2007), técnicas de visualização em paisagens 3D tentam apresentar hierarquias usando uma metáfora de paisagem 3D. Para ontologias, esse tipo de técnica poderia ser útil fornecendo uma dimensão extra onde as propriedades do nó poderiam ser codificadas e os links de relação apresentados (KATIFORI *et al.*, 2007).

Segundo Katifori *et al.* (2007), existiam dois sistemas nessa categoria para apresentação de hierarquias, porém nenhum dos dois foi aplicado para ontologias. Os autores ainda destacam que não é claro se esse tipo de visualização poderia ser útil no contexto de ontologias. Mas destacam que esse tipo pode ser eficaz para tarefas de visão geral de hierarquia, desde que combinado com ferramentas adequadas de busca e filtragem, e mecanismos de navegação intuitivos, simples e eficazes.

4.7 Discussão da aplicação das categorias de visualização para anomalias

Nas últimas 6 seções, foram apresentadas diferentes formas de visualizar ontologias. Essa análise é importante para definir qual tipo de método é melhor para visualizar uma anomalia encontrada em ontologias. Dentre todas as categorias, a que mais se adéqua a visualização de anomalias é a *Node-link and tree*. Para tirar essa conclusão, foram levadas em consideração as vantagens e desvantagens de todas as categorias.

A grande vantagem da categoria *Node-link and tree* é que árvores e grafos são uma maneira comum e intuitiva de representação de hierarquias. Pensando que a visualização de uma anomalia também necessita ser o mais intuitiva possível, de forma a facilitar o seu entendimento, esse tipo de técnica se torna uma ótima opção. Outro ponto importante na escolha desse tipo de técnica é que sua desvantagem (uso ineficiente do espaço de tela) não é relevante para visualizar apenas uma ocorrência de anomalia por vez.

A categoria *Indented list* é descartada por apenas representar heranças taxonômicas. Porém, nas anomalias existem outros tipos de relações entre entidades que são necessários para o entendimento da anomalia, como apresentado no capítulo anterior.

As categorias *Zoomable*, *Space-filling* e *Focus + context or distortion* são descartadas por se tratarem de técnicas voltadas para visualização da ontologia completa. Apenas

as estruturas envolvidas na anomalia serão visualizada para que não se gere confusão ao desenvolvedor da ontologia.

A categoria *3D Information landscapes* é descartada por ainda não se ter uma aplicação para a visualização de ontologias e por ser um modo de representar grandes quantidades de informações. No caso de anomalias, a aplicação desse tipo de técnica não parece ser a mais eficiente.

4.8 Considerações finais

Ontologias se tornaram um meio importante para facilitar a estruturação e a organização de informações. Porém, muitas delas se tornaram bastante complexas. Isso tornou a compreensão e correção de anomalias um processo mais difícil. Uma possibilidade para facilitar o entendimento de anomalias é visualizá-las graficamente. Esse entendimento pode auxiliar o desenvolvedor da ontologia na tomada de decisão sobre quais alterações são necessárias para a correção de uma anomalia.

Neste capítulo foram apresentados os principais métodos utilizados para visualizar ontologias. Foram apontadas as principais vantagens e desvantagens desses métodos e, por último, foi apresentada a decisão de escolha pela categoria *Node-link and tree* de visualização para ser implementada na visualização proposta neste trabalho.

As implementações da visualização e do método de identificação de anomalias, propostos neste trabalho, serão apresentadas no próximo capítulo.

MÉTODO ONTO-ANALYST PARA IDENTIFICAÇÃO E VISUALIZAÇÃO DE ANOMALIAS

O objetivo principal deste trabalho foi propor um método para identificar ocorrências de anomalias em ontologias, chamado ONTO-Analyst, que permite a desenvolvedores descrever essas anomalias usando consultas SPARQL e visualizá-las de forma gráfica. Foi proposta também a implementação e o uso de uma ontologia, a *METADATA description For Ontologies/Rules* (MetaFOR), para descrever a estrutura de outra ontologia e usar consultas SPARQL para identificar anomalias nessa descrição. A proposta também prevê que, uma vez identificadas, as anomalias podem ser apresentadas de forma visual em grafos com nós representando as entidades envolvidas na anomalia e as arestas representando as relações entre elas.

O método ONTO-Analyst criado nesta tese é apresentado por meio do fluxograma encontrado na Figura 14. Inicialmente o método ONTO-Analyst carrega uma ontologia a ser analisada e realiza uma verificação dessa ontologia com um *reasoner* (Pellet ou Hermit). Caso existam erros na ontologia analisada, esses são mostrados ao usuário. Na sequência, o método extrai informações relevantes para identificação de anomalias da ontologia analisada e as insere em uma ontologia no formato MetaFOR, formato que será descrito posteriormente neste capítulo. Com essa ontologia no formato MetaFOR, podem ser executadas consultas SPARQL que retornam uma lista de anomalias da ontologia analisada. As consultas podem ser editadas, criadas e executadas interativamente. Para cada anomalia encontrada pela consulta SPARQL pode ser gerada uma visualização que mostra o contexto dessa anomalia.

Para apoiar o uso desse método, um protótipo de sistema, o ONTO-Analyst, foi desenvolvido, de forma a demonstrar a sua eficácia. Como descrito, o protótipo lê uma

Figura 14 – Fluxograma mostrando o processo do método ONTO-Analyst.

Fonte: Elaborada pelo autor.

ontologia em OWL, cria uma descrição da sua estrutura usando a ontologia MetaFOR, aplica um conjunto de consultas SPARQL para encontrar anomalias, e gera uma visualização para as anomalias encontradas. Como o ONTO-Analyst cria uma descrição da estrutura da ontologia e usa a linguagem SPARQL, uma linguagem popular entre desenvolvedores de ontologias, ele permite que os próprios desenvolvedores escrevam consultas para encontrar novos tipos de anomalias. Regras em SWRL podem ser usadas no lugar de consultas SPARQL para encontrar anomalias. A escolha fica por conta do usuário.

A identificação de anomalias usando uma ontologia (MetaFOR) traz três importantes vantagens aos desenvolvedores de ontologias:

- A facilidade de utilizar recursos que eles conhecem, tais como: ontologias, consultas SPARQL e regras SWRL;
- A possibilidade de alterar os critérios usados na identificação de anomalias, caso necessário. Como essa identificação é feita por consultas SPARQL e/ou regras SWRL, eles poderão fazer alterações nessas consultas sem dificuldade, pois elas são lingua-

gens bem conhecidas na comunidade da Web Semântica;

- A facilidade de adicionar a identificação de outros tipos de anomalias, ou qualquer outro tipo de padrão estrutural que eles considerarem importantes, na descrição estrutural da ontologia (no formato MetaFOR). Para isso, eles deverão apenas escrever novas consultas em SPARQL ou SWRL.

Outro ponto importante do método é a visualização que funciona em colaboração com a identificação. Para cada identificação de anomalia escrita em uma consulta SPARQL, o sistema ONTO-Analyst gera uma visualização com o intuito de mostrar somente as partes envolvidas na anomalia. Essa é uma importante característica, pois, normalmente, as visualizações de ontologias são gerais e apresentam todas as partes da ontologia. Ao visualizar somente os elementos que fazem parte da anomalia, o desenvolvedor pode focar direto na solução da mesma.

A apresentação do método ONTO-Analyst será dividida em:

- O formato para a descrição da estrutura de ontologias, que contém os metadados extraídos das ontologias a serem analisadas na identificação de anomalias. Esse formato usa a ontologia MetaFOR e será apresentado na Seção 5.1;
- O protótipo do sistema ONTO-Analyst para identificação e visualização de anomalias. O funcionamento básico desse protótipo será apresentado na Seção 5.2;
- As consultas SPARQL elaboradas para a identificação de anomalias, disponibilizadas por padrão no ONTO-Analyst. Essas consultas serão apresentadas na subseção 5.3;
- Os detalhes gerais da visualização das anomalias encontradas (com as consultas SPARQL). Esses detalhes são descritos na subseção 5.4.

5.1 Ontologia MetaFOR

A ontologia *METAdata description For Ontologies/Rules* (MetaFOR) é a usada para representar a estrutura das ontologias a serem analisadas pelo método ONTO-Analyst. O formato MetaFOR descreve a estrutura de uma ontologia em OWL usando instâncias das classes e propriedades, definidas na ontologia MetaFOR, para representar as classes e propriedades dessa ontologia. Esse formato representa partes da estrutura da ontologia analisada para que anomalias (ou outros padrões estruturais) possam ser identificadas por meio de consultas SPARQL (ou regras SWRL). Para que essas identificações sejam possíveis, no formato MetaFOR são representados três tipos de informações extraídas das ontologias analisadas:

- Entidades são representadas como instâncias de classes da ontologia MetaFOR;
- Relações entre entidades são representadas como instâncias de propriedades de objetos da ontologia MetaFOR;
- Valores de entidades são representados como instâncias de propriedades de dados da ontologia MetaFOR.

Esse formato de representação foi adotado, porque o domínio da ontologia MetaFOR são as estruturas das ontologias analisadas. Por isso, essas estruturas são representadas como instâncias de classes ou propriedades da ontologia MetaFOR. Cada um dos tipos de informação extraída das ontologias será apresentado nas próximas seções.

5.1.1 Representação de entidades

As entidades das ontologias analisadas são representadas como instâncias de classes na ontologia MetaFOR. Para isso, foram criadas classes na MetaFOR para cada tipo de entidade encontrada numa ontologia:

- **Class**: classes são instanciadas usando-se a classe **Class**;
- **ObjectProperty**: propriedades de objetos são instanciadas usando-se a classe **ObjectProperty**. Essa classe possui sete subclasses para diferentes tipos de propriedades de objeto:
 - **TransitiveObjectProperty**: instâncias de propriedades transitivas;
 - **SymmetricObjectProperty**: instâncias de propriedades simétricas;
 - **FunctionalObjectProperty**: instâncias de propriedades funcionais;
 - **InverseFunctionalObjectProperty**: instâncias de propriedades inversamente funcionais;
 - **AsymmetricObjectProperty**: instâncias de propriedades assimétricas;
 - **ReflexiveObjectProperty**: instâncias de propriedades reflexivas;
 - **IrreflexiveObjectProperty**: instâncias de propriedades não reflexivas.
- **DataProperty**: propriedades de dados são instanciadas usando-se a classe **DataProperty**. Essa classe tem uma subclasse:
 - **FunctionalDataProperty**: instâncias de propriedades funcionais.
- **Rule**: regras são instanciadas usando-se a classe **Rule**;
- **Atom**: átomos de regras são instanciados usando-se a classe **Atom**;

- **Argument**: argumentos de átomos são instanciados usando-se a classe **Argument**.
- **Cardinality**: representações de cardinalidades são instanciadas usando-se a classe **Cardinality**. Essa classe possui cinco subclasses para diferentes tipos de cardinalidades:
 - **MaxCardinality**: instâncias de cardinalidades do tipo *maxCardinality*;
 - **MinCardinality**: instâncias de cardinalidades do tipo *minCardinality*;
 - **ExactCardinality**: instâncias de cardinalidades do tipo *exactCardinality*;
 - **SomeValuesFromCardinality**: instâncias de cardinalidades do tipo *someValuesFrom*;
 - **AllValuesFromCardinality**: instâncias de cardinalidades do tipo *allValuesFrom*.

Figura 15 – A propriedade de dados **has_current_age** (esquerda, em verde) X sua representação no formato MetaFOR (direita, em laranja).

Fonte: Elaborada pelo autor.

Um exemplo de extração de informações de uma ontologia e inserção dessas informações em uma ontologia no formato MetaFOR pode ser visto na Figura 15. Na parte esquerda da figura é mostrada uma propriedade de dados (**has_current_age**) de uma ontologia que está sendo analisada e, na parte direita, é mostrada essa mesma propriedade convertida para o formato MetaFOR: ela se tornou uma instância, **URI1_has_current_age**, da classe **DataProperty**. O nome da entidade em MetaFOR é similar ao nome original na ontologia analisada, com a adição de um prefixo para informar de qual ontologia aquela entidade foi extraída. Isso é necessário, devido a casos em que possam existir entidades com nomes iguais de diferentes ontologias integradas na ontologia analisada.

5.1.2 Representações de relações entre entidades

As relações entre entidades das ontologias analisadas são representadas como instâncias de propriedades de objetos na ontologia MetaFOR. Para isso, foram criadas diversas propriedades de objeto na MetaFOR que representam essas relações:

- **hasCardinality**: relaciona uma entidade com uma restrição de cardinalidade;
- **isDisjoint**: relaciona classes ou propriedades disjuntas;
- **hasDomain**: relaciona uma propriedade ao seu `rdfs:domain`;
- **hasRange**: relaciona uma propriedade ao seu `rdfs:range`;
- **isEquivalent**: relaciona classes ou propriedades equivalentes;
- **hasInverse**: relaciona classes ou propriedades inversas;
- **hasSuper**: relaciona uma subentidade (subclasse ou subpropriedade) com cada uma das suas superentidades diretas ou indiretas;
- **hasDirectSuper**: relaciona uma subentidade com cada uma das suas superentidades diretas;
- **hasSub**: propriedade inversa de **hasSuper**;
- **hasDirectSub**: propriedade inversa de **hasDirectSuper**;
- **hasAntecedentAtom**, **hasConsequentAtom**: relacionam uma regra com cada um dos seus átomos do antecedente ou do consequente;
- **hasPredicate**: relaciona um átomo ao seu predicado;
- **hasArgument**, **hasFirstArgument**, **hasSecondArgument**: relacionam um átomo com cada um dos seus argumentos. **hasFirstArgument** e **hasSecondArgument** são subpropriedades de **hasArgument** e representam, respectivamente, o primeiro e o segundo argumento de um átomo. Por exemplo, no átomo `hasParent(?X, ?Y)`, `?X` é o primeiro argumento e `?Y` é o segundo argumento;
- **sameAntecedent**, **sameConsequent**: relacionam duas regras declarando que elas tem, respectivamente, os mesmos antecedentes e os mesmos consequentes. Significa que cada átomo do antecedente (ou consequente) de uma regra é igual ou equivalente a outro átomo no antecedente (ou consequente) da outra regra;
- **subsumes**: relaciona duas regras em que o antecedente da primeira regra, subsume (engloba) o antecedente da segunda regra. Para uma regra A subsumir uma regra B, todos os átomos do antecedente da regra B devem estar presentes no antecedente

da regra A. Além disso, o sistema ONTO-Analyst também pode encontrar casos de subsunção que usem built-ins como `greaterThanOrEqual`, `lessThanOrEqual`, entre outros.

Figura 16 – Representação no formato MetaFOR da propriedade `has_adoptive_daughter` e suas relações com outras entidades da ontologia original.

Fonte: Elaborada pelo autor.

Um exemplo da representação das relações entre entidades no formato MetaFOR pode ser visto na Figura 16. Nessa figura, a entidade `URI1_has_adoptive_daughter`, uma propriedade de objeto na ontologia analisada, é representada no formato MetaFOR como uma instância da classe `ObjectProperty` e possui cinco relações com outras instâncias (que também representam classes e propriedades na ontologia original). Essas relações são compostas por três relações de super propriedades (`hasSuper`), uma relação direta de super propriedade (`hasDirectSuper`) e uma relação de `rdfs:range` (`hasRange`).

5.1.3 Representações de relações entre entidades e valores

As relações entre entidades das ontologias analisadas e seus valores são representadas como instâncias de propriedades de dados na ontologia MetaFOR. Para isso, foram criadas diversas propriedades de dados na MetaFOR que representam determinadas características de entidades:

- `hasCardinalityNumber`: valores numéricos para instâncias da classe `Cardinality`;
- `hasInstancesNumber`: número total de instâncias de uma determinada classe;
- `isLeaf`: propriedade booleana que declara que uma entidade não tem entidades filhas. Por exemplo, se uma classe não tem subclasses, a instância, que a representa no formato MetaFOR, possui `true` para a propriedade `isLeaf`;

- **usedInRestrictions**: propriedade booleana que declara se uma entidade é utilizada em restrições;
- **usedInRules**: propriedade booleana que declara se uma entidade é utilizada em, pelo menos, uma regra.

Figura 17 – Representação de propriedades de dados de entidades da ontologia analisada no formato MetaFOR.

Fonte: Elaborada pelo autor.

Um exemplo da representação de características das entidades (de uma ontologia analisada), no formato MetaFOR, pode ser visto na Figura 17. Nessa figura, a classe *At_Age_20_to_29* (da ontologia analisada) é representada (no formato MetaFOR) pela instância *URI1_At_Age_20_to_29* da classe *Class* e possui quatro propriedades de dados: *hasInstancesNumber* igual a 1, *isLeaf* igual a *true*, *usedInRules* e *usedInRestrictions* iguais a *false*.

5.1.4 Extensibilidade

A MetaFOR pode representar diversos aspectos da estrutura de uma ontologia. Contudo, pode ser que algum aspecto de interesse de um determinado grupo de desenvolvedores não seja contemplado. Nesse caso, não seria possível escrever consultas SPARQL (ou regras SWRL) que incluíssem esse aspecto. Para resolver esse problema foi implementado, no protótipo ONTO-Analyst, uma forma de extrair outras informações das ontologias analisadas e inserir no formato MetaFOR. Para isso, são usadas interfaces Java e um carregador automático de implementações no ONTO-Analyst. Dessa forma, basta criar novas implementações das interfaces Java e adicionar essa implementação, em forma de arquivo JAR, que o ONTO-Analyst extrairá essa nova informação. Mais detalhes desse funcionamento serão apresentados na Seção 5.2.1.

5.2 Protótipo do sistema ONTO-Analyst

Nesta subseção são mostrados os detalhes de implementação do protótipo ONTO-Analyst e sua relação com a ontologia MetaFOR. O protótipo do sistema ONTO-Analyst foi desenvolvido em Java. Sua tela principal pode ser vista na Figura 18.

Figura 18 – Tela principal do protótipo ONTO-Analyst: Ele tem um painel editor para consultas SPARQL, uma tabela para listar resultados das consultas e um painel para a representação das visualizações

Fonte: Elaborada pelo autor.

O sistema ONTO-Analyst converte ontologias OWL para o formato MetaFOR, aplica um conjunto de consultas SPARQL e mostra as anomalias encontradas. Ele tem menus *drop-down* em que usuários podem carregar ontologias a serem analisadas ou exportar ontologias no formato MetaFOR (Menu *MetaFOR Ontology*). Além disso, os usuários podem editar e salvar consultas SPARQL ou carregá-las de arquivos (Menu *SPARQL File*). Quando uma consulta é carregada, ela aparece na caixa de texto (Figura 18, no lado esquerdo superior). A consulta SPARQL pode ser editada nessa caixa de texto e executada usando o botão *Execute query*. Com esse modo de operação, desenvolvedores de ontologias podem escrever e testar suas próprias consultas. Essas consultas podem detectar novas anomalias ou detectar quebras de padrões de projeto específicos às ontologias que eles estão construindo.

Após executar uma identificação de anomalia (usando uma consulta SPARQL), seus resultados podem ser vistos na tabela, abaixo do botão *Execute query*. Com esses resultados, o usuário pode selecionar uma ocorrência dessa anomalia para exibição no painel de visualização gráfica (Figura 18, lado direito).

Internamente, o sistema foi planejado para funcionar em três etapas (Figura 19). A primeira é responsável por carregar as informações necessárias para que a conversão da ontologia seja executada. Essa etapa usa os quatro componentes, mostrados no diagrama de componentes apresentado na Figura 19: *OntologyLoader*, *EntityLoader*,

`RelationLoader` e `URIManager`. A segunda etapa realiza a conversão para o formato MetaFOR e usa o componente `Converter` e seus sub componentes. A terceira etapa executa as consultas em SPARQL para a identificação das ocorrências de anomalias (`AnomalyFinder`) e gera as visualizações necessárias para as anomalias encontradas (`AnomalyViewer`). Opcionalmente, é possível exportar a ontologia gerada (componente `Exporter`) no formato MetaFOR para análise por outras ferramentas.

Figura 19 – Diagrama de componentes do ONTO-Analyst com suas interfaces.

Fonte: Elaborada pelo autor.

Cada um dos estágios apresentados na Figura 19 será apresentado com mais detalhes nas próximas três subseções.

5.2.1 Estágio 1: Preparação da conversão da ontologia analisada para o formato MetaFOR

No primeiro estágio, o sistema ONTO-Analyst carrega todas as informações necessárias para a conversão de uma ontologia analisada para o formato MetaFOR. O estágio se inicia com o componente `OntologyLoader` que instancia objetos da OWL API necessários nos próximos dois estágios. A OWL API (versão 3.5.2) suporta a criação, manipulação e serialização de ontologias OWL (HORRIDGE; BECHHOFFER, 2011). É utilizada no sistema ONTO-Analyst para manipular ontologias em todos os estágios da conversão,

através do componente `OntologyLoader`. A OWL API pode carregar ontologias em seis diferentes formatos de arquivos: RDF/XML, OWL/XML, OWL Functional Syntax, Turtle, KRSS e OBO Flat. Portanto, todos esses formatos de arquivo são suportados pelo sistema ONTO-Analyst. De um modo geral, o objetivo do `OntologyLoader` é separar as chamadas OWL API do resto do sistema. Essa separação permite, caso seja necessário, a fácil substituição da OWL API por outra API.

Além desses componentes, o `URIManager` também atua nesse estágio para gerenciar as diversas URIs encontradas em uma ontologia analisada. Esse é um componente simples que gerencia as diferentes URIs bases das entidades analisadas. Por exemplo, se existirem duas ontologias integradas que apresentam a entidade `Person`, esse componente se encarregará de gerenciar essa integração dentro do formato MetaFOR com a criação das instâncias `URI1_Person` e `URI2_Person` dentro da ontologia MetaFOR. É importante destacar a necessidade desse componente, pois não se pode usar a URI base de uma entidade extraída de uma ontologia analisada, pois essa entidade é uma entidade na ontologia analisada e não poderia ser representada com a mesma URI base no formato MetaFOR.

Os dois últimos componentes desse estágio são os componentes `EntityLoader` e `RelationLoader`, que são responsáveis por carregar as implementações de interfaces java das interfaces `EntityInterface` e `RelationInterface`, respectivamente. As classes que implementam essas interfaces são utilizadas para criar as estruturas das ontologias no formato MetaFOR. Por exemplo, como apresentado na Subseção 5.1.1, existem algumas implementações básicas de conversão das entidades da ontologia analisada para o formato MetaFOR, como `Class`, `ObjectProperty`, `DataProperty`, `Rule`, etc. Para todas essas entidades, foram criadas classes que implementam a interface java `EntityInterface`. Essas implementações contêm métodos Java que criam as estruturas e as instâncias no formato MetaFOR. O mesmo se aplica para a interface `RelationInterface`, que é responsável por criar os tipos de relações e as instâncias de relações no formato MetaFOR (os tipos de relações criados podem ser vistos nas Subseções 5.1.2 e 5.1.3).

Para carregar de forma automática as implementações das interfaces `EntityInterface` e `RelationInterface`, foi usada a ferramenta `ServiceLoader`¹ nos componentes `EntityLoader` e `RelationLoader`. Essa ferramenta foi utilizada para carregar novas implementações Java em arquivos JAR. Sua maior vantagem é possibilitar que usuários (avançados do sistema) adicionem novas implementações dessas interfaces mais facilmente, sem a necessidade de alterar todo o sistema. Por exemplo, se algum usuário necessitar que o formato MetaFOR tenha uma subclassificação de classes chamada "Classes Anônimas", ele pode implementar todos os métodos da Interface `EntityInterface`, gerar um arquivo JAR com essa implementação e adicionar esse arquivo na mesma pasta do sistema. Na próxima vez que o sistema rodar, ele terá a possibilidade de extrair informações sobre essa nova entidade.

¹ docs.oracle.com/javase/7/docs/api/java/util/ServiceLoader.html

Ao final desse estágio, todas as informações necessárias para iniciar a conversão da ontologia analisada para o formato MetaFOR estão carregadas.

5.2.2 Estágio 2: Conversão da ontologia analisada para o formato MetaFOR

No segundo estágio, o componente `Converter` e seus quatro subcomponentes são utilizados (Figura 19). Esses componentes realizam a conversão de uma ontologia analisada para o formato MetaFOR.

Os componentes `EntityCreator` e `RelationCreator` instanciam a ontologia MetaFOR, criando suas classes, propriedades de objetos e propriedades de dados. Esses componentes utilizam as classes em Java carregadas pelos componentes `EntitiesLoader` e `RelationLoader` no estágio anterior.

Após a construção da estrutura básica da MetaFOR, a conversão pode ser iniciada. O componente `IndividualExtractor` cria todas as instâncias de classes da ontologia MetaFOR para representar classes, propriedades, regras, etc da ontologia que está sendo convertida. Para cada instância criada no formato MetaFOR, o componente `RelationExtractor` cria todas as relações dessa instância, com base nas informações encontradas na ontologia analisada.

Por exemplo, considere que uma ontologia analisada tem uma propriedade de objeto `hasParent` com uma restrição `rdfs:domain` usando a classe `Person` (apenas instâncias de `Person` podem ter essa propriedade). O componente `IndividualExtractor` cria as instâncias `URI1_Person`, da classe `Classes`, e `URI1_hasParent`, da classe `ObjectProperty`. Depois, no componente `RelationExtractor`, a relação `hasDomain` é definida entre as instâncias `URI1_hasParent` e `URI1_Person`.

Ao final desse estágio, uma ontologia no formato MetaFOR foi criada com todas as informações relevantes da ontologia analisada. Com a ontologia no formato MetaFOR é possível iniciar a execução das identificações de anomalias.

5.2.3 Estágio 3: Operações com uma ontologia no formato MetaFOR

Após os componentes `IndividualExtractor` e `RelationExtractor` terem convertido todas as entidades da ontologia analisada, a ontologia no formato MetaFOR é analisada pelo componente `AnomalyFinder`. Opcionalmente, essa ontologia também pode ser salva pelo componente `Exporter`. Ele permite que ontologias no formato MetaFOR possam ser examinadas em editores de ontologias, como o Protégé.

O componente `AnomalyFinder` carrega e executa consultas SPARQL para identificar ocorrências de anomalias. Ele também permite que usuários editem as consultas,

mudando o modo como uma anomalia é encontrada ou criando novas consultas para identificar novas anomalias. Além disso, nesse estágio é possível visualizar as anomalias encontradas (com as consultas SPARQL) por meio do componente `AnomalyViewer`.

5.3 Consultas SPARQL para identificação das anomalias no formato MetaFOR

A partir da seleção das dezoito anomalias apresentadas na Seção 3.4, foram elaboradas as consultas SPARQL que identificam essas anomalias em ontologias no formato MetaFOR. Nesta seção, todas essas consultas, disponibilizadas por default no ONTO-Analyst, são apresentadas. De modo a não repetir os prefixos em todas as consultas SPARQL, o Código-fonte 11 apresenta os dois prefixos usados em todas as consultas desta seção.

Código-fonte 11 – Prefixos padrões usados nas consultas SPARQL que identificam anomalias

```
1: @prefix metaFor: <www.metaontology.org/ontologies/metaontology#> .  
2: @prefix rdf: <www.w3.org/1999/02/22-rdf-syntax-ns#> .
```

Fonte: Elaborada pelo autor.

Para cada uma das anomalias é apresentada a consulta SPARQL que faz a sua identificação em ontologias no formato MetaFOR e uma descrição geral dessa consulta. A apresentação das anomalias será feita agrupando-as em três categorias: anomalias de classes/propriedades, de cardinalidades e de regras.

5.3.1 Anomalias que ocorrem em classes e propriedades

Como apresentado na Seção 3.4, existem sete anomalias selecionadas que ocorrem em classes e/ou propriedades.

A consulta para identificação da anomalia *Chains of inheritance* pode ser vista no Código-fonte 12. Ela identifica conjuntos de três classes que estejam em uma hierarquia direta em que nenhuma das classes contenha instâncias e nenhuma delas seja utilizada em restrições ou regras SWRL.

Código-fonte 12 – Identificação da anomalia *Chains of inheritance* usando SPARQL

```
1: SELECT DISTINCT ?c2  
2: WHERE { ?c1 rdf:type metaFor:Class ;  
3:           metaFor:hasInstancesNumber 0 ;
```

```

4:          metaFor:usedInRestrictions false ;
5:          metaFor:usedInRules false ;
6:          metaFor:hasDirectSub ?c2 .
7:      ?c2 metaFor:hasInstancesNumber 0 ;
8:          metaFor:usedInRestrictions false ;
9:          metaFor:usedInRules false ;
10:         metaFor:hasDirectSub ?c3 .
11:      ?c3 metaFor:hasInstancesNumber 0 ;
12:          metaFor:usedInRestrictions false ;
13:          metaFor:usedInRules false      }

```

Fonte: Elaborada pelo autor.

A consulta para identificação da anomalia *Lonely disjoint classes* pode ser vista no Código-fonte 13. Ela identifica conjuntos de três classes disjuntas que não pertencem ao mesmo ramo da taxonomia.

Código-fonte 13 – Identificação da anomalia *Lonely disjoint classes* usando SPARQL

```

1: SELECT ?ca1
2: WHERE { ?ca1 rdf:type metaFor:Class ;
3:          metaFor:isDisjoint ?cb1 ;
4:          metaFor:isDisjoint ?cb2 ;
5:          metaFor:hasDirectSuper ?superA .
6:      ?cb1 metaFor:isDisjoint ?cb2 ;
7:          metaFor:hasDirectSuper ?superB .
8:      ?cb2 metaFor:hasDirectSuper ?superB
9:      FILTER (!(?superA = ?superB))

```

Fonte: Elaborada pelo autor.

A consulta para identificação da anomalia *Partition error in taxonomy* pode ser vista no Código-fonte 14. Ela identifica classes que possuem duas superclasses disjuntas.

Código-fonte 14 – Identificação da anomalia *Partition error in taxonomy* usando SPARQL

```

1: SELECT DISTINCT ?cf
2: WHERE { ?c1 metaFor:isDisjoint ?c2 .
3:         ?c1 metaFor:hasSub ?cf .
4:         ?c2 metaFor:hasSub ?cf }

```

Fonte: Elaborada pelo autor.

A consulta para identificação da anomalia *Exact circularity in taxonomy* pode ser vista no Código-fonte 15. Ela identifica conjuntos de quatro classes ou quatro propriedades que possuem um ciclo entre elas por meio de definições de heranças e equivalências.

Código-fonte 15 – Identificação da anomalia *Exact circularity in taxonomy* usando SPARQL

```
1: SELECT ?clsA ?clsB ?subA ?superB
2: WHERE { ?clsA metaFor:isEquivalent ?clsB .
3:         ?clsA metaFor:hasSub ?subA .
4:         ?subA metaFor:isEquivalent ?superB .
5:         ?clsB metaFor:hasSuper ?superB }
```

Fonte: Elaborada pelo autor.

A consulta para identificação da anomalia *Redundancy by repetitive taxonomic definition* pode ser vista no Código-fonte 16. Ela identifica conjuntos de três classes ou três propriedades que possuem uma definição redundante de herança.

Código-fonte 16 – Identificação da anomalia *Redundancy by repetitive taxonomic definition* usando SPARQL

```
1: SELECT DISTINCT ?clsC
2: WHERE { ?clsA metaFor:hasDirectSub ?clsB .
3:         ?clsB metaFor:hasDirectSub ?clsC .
4:         ?clsA metaFor:hasDirectSub ?clsC }
```

Fonte: Elaborada pelo autor.

A consulta para identificação da anomalia *Lazy class or property* pode ser vista no Código-fonte 17. Ela identifica classes ou propriedades que são folhas na hierarquia, não possuem instâncias e não são usadas em restrições ou regras.

Código-fonte 17 – Identificação da anomalia *Lazy class or property* usando SPARQL

```
1: SELECT ?s
2: WHERE { ?s metaFor:isLeaf true ;
3:         metaFor:usedInRestrictions false ;
4:         metaFor:usedInRules false ;
5:         metaFor:hasInstancesNumber 0 }
```

Fonte: Elaborada pelo autor.

A consulta para identificação da anomalia *Circular properties* pode ser vista no Código-fonte 18. Ela identifica conjuntos de duas propriedades inversas que possuem as definições para o mesmo `rdfs:domain` e para o mesmo `rdfs:range`.

Código-fonte 18 – Identificação da anomalia *Circular properties* usando SPARQL

```

1:
2: SELECT ?p1 ?p2
3: WHERE { ?p1 metaFor:hasInverse ?p2 ;
4:         metaFor:hasDomain ?d1 ;
5:         metaFor:hasRange ?r1 .
6:         ?p2 metaFor:hasDomain ?d2 ;
7:         metaFor:hasRange ?r2 . ?d1
8:         metaFor:isEquivalent ?d2 .
9:         ?r1 metaFor:isEquivalent ?r2
10:        FILTER (?d1 != ?r1) }
```

Fonte: Elaborada pelo autor.

5.3.2 Anomalias que ocorrem em cardinalidades

Como apresentado na Seção 3.4, existem duas anomalias que ocorrem em definições de cardinalidades.

A consulta para identificação da anomalia *Multiple functional properties* pode ser vista no Código-fonte 19. Ela identifica propriedades funcionais que apresentam uma definição inconsistente de cardinalidade máxima.

Código-fonte 19 – Identificação da anomalia *Multiple functional properties* usando SPARQL

```

1: SELECT DISTINCT ?p
2: WHERE { ?p rdf:type metaFor:FunctionalObjectProperty ;
3:         metaFor:hasCardinality ?c1 .
4:         ?c1 rdf:type metaFor:MaxCardinality ;
5:         metaFor:hasCardinalityNumber ?maxcard
6:         FILTER (?maxcard > 1) }
```

Fonte: Elaborada pelo autor.

Foram apresentadas duas formas de ocorrência da anomalia *Redundant cardinalities*, com duas definições de redundâncias em cardinalidades diferentes. Por isso, foi necessária a elaboração de duas consultas SPARQL que podem ser vista nos Códigos-fontes 20

e 21. A primeira identifica o uso redundante da definição de cardinalidade mínima e a segunda identifica o uso redundante da definição de cardinalidade máxima em propriedades que tenham uma super propriedade funcional.

Código-fonte 20 – Identificação do primeiro caso da anomalia *Redundant cardinalities* usando SPARQL

```
1: SELECT DISTINCT ?property
2: WHERE { ?property metaFor:hasCardinality ?c .
3:         ?c rdf:type metaFor:MinCardinality ;
4:         metaFor:hasCardinalityNumber 0 }
```

Fonte: Elaborada pelo autor.

Código-fonte 21 – Identificação do segundo caso da anomalia *Redundant cardinalities* usando SPARQL

```
1: SELECT DISTINCT ?p1 ?p2
2: WHERE { ?p2 metaFor:hasCardinality ?c ;
3:         metaFor:hasSuper ?p1 .
4:         ?c rdf:type metaFor:MaxCardinality ;
5:         metaFor:hasCardinalityNumber 1 .
6:         ?p1 rdf:type metaFor:FunctionalObjectProperty }
```

Fonte: Elaborada pelo autor.

5.3.3 Anomalias que ocorrem em regras SWRL

Como apresentado na Seção 3.4, existem nove anomalias que ocorrem em regras SWRL.

A consulta para identificação da anomalia *Circularity between rules and taxonomy* pode ser vista no Código-fonte 22. Essa consulta identifica regras em que um átomo do consequente implica em um átomo do antecedente por meio dos seus predicados.

Código-fonte 22 – Identificação da anomalia *Circularity between rules and taxonomy* usando SPARQL

```
1: SELECT DISTINCT ?rule
2: WHERE { ?rule rdf:type metaFor:Rule ;
3:         metaFor:hasAntecedentAtom ?laa ;
4:         metaFor:hasConsequentAtom ?lca .
5:         ?laa metaFor:hasPredicate ?p1 .
```

```

6:      ?lca metaFor:hasPredicate ?p2 .
7:      ?p1 rdf:type metaFor:Class ;
8:      metaFor:hasSub ?p2 }

```

Fonte: Elaborada pelo autor.

A consulta para identificação da anomalia *Contradicting rules* pode ser vista no Código-fonte 23. Essa consulta identifica conjuntos de duas regras que possuem os mesmos antecedentes, porém os consequentes apresentam contradições.

Código-fonte 23 – Identificação da anomalia *Contradicting rules* usando SPARQL

```

1: SELECT DISTINCT ?rule1
2: WHERE { ?rule1 metaFor:sameAntecedent ?rule2 ;
3:         metaFor:hasConsequentAtom ?ca1 .
4:         ?rule2 metaFor:hasConsequentAtom ?ca2 .
5:         ?ca1 metaFor:hasPredicate ?p1 .
6:         ?ca2 metaFor:hasPredicate ?p2 .
7:         ?p1 metaFor:isDisjoint ?p2 }

```

Fonte: Elaborada pelo autor.

Para realizar a identificação da anomalia *Incompatible rule antecedent* foi decidido separar a consulta SPARQL em duas, de modo a simplificar o entendimento da consulta utilizada na identificação dessa anomalia. Foi criada uma consulta para regras SWRL em que a incompatibilidade ocorre por meio de classes disjuntas (Código-fonte 24) e outra consulta em que a incompatibilidade ocorre por meio de propriedades de objeto disjuntas (Código-fonte 25).

Código-fonte 24 – Identificação da anomalia *Incompatible rule antecedent* em classes usando SPARQL

```

1: SELECT DISTINCT ?rule
2: WHERE { ?rule metaFor:hasAntecedentAtom ?a1 ;
3:         metaFor:hasAntecedentAtom ?a2 .
4:         ?a1 metaFor:hasPredicate ?p1 ;
5:         metaFor:hasFirstArgument ?var1 .
6:         ?a2 metaFor:hasPredicate ?p2 ;
7:         metaFor:hasFirstArgument ?var1 .
8:         ?p1 metaFor:isDisjoint ?p2 .
9:         ?p1 rdf:type metaFor:Class .
10:        ?p2 rdf:type metaFor:Class }

```

Fonte: Elaborada pelo autor.

Código-fonte 25 – Identificação da anomalia *Incompatible rule antecedent* em propriedades de objetos usando SPARQL

```

1: SELECT DISTINCT ?rule
2: WHERE { ?rule metaFor:hasAntecedentAtom ?a1 ;
3:         metaFor:hasAntecedentAtom ?a2 .
4:         ?a1 metaFor:hasPredicate ?p1 ;
5:         metaFor:hasFirstArgument ?var1 ;
6:         metaFor:hasSecondArgument ?var2 .
7:         ?a2 metaFor:hasPredicate ?p2 ;
8:         metaFor:hasFirstArgument ?var1 ;
9:         metaFor:hasSecondArgument ?var2 .
10:        ?p1 metaFor:isDisjoint ?p2 }

```

Fonte: Elaborada pelo autor.

A consulta para identificação da anomalia *Redundant derivation in the antecedent* pode ser vista no Código-fonte 26. Ela identifica regras que possuem dois átomos do antecedente com predicados que são uma classe e a sua subclasse.

Código-fonte 26 – Identificação da anomalia *Redundant derivation in the antecedent* usando SPARQL

```

1: SELECT DISTINCT ?rule
2: WHERE { ?rule metaFor:hasAntecedentAtom ?laa1 ;
3:         metaFor:hasAntecedentAtom ?laa2 .
4:         ?laa1 metaFor:hasPredicate ?p1 ;
5:         metaFor:hasFirstArgument ?arg1 .
6:         ?laa2 metaFor:hasPredicate ?p2 ;
7:         metaFor:hasFirstArgument ?arg1 .
8:         ?p1 rdf:type metaFor:Class ;
9:         metaFor:hasSub ?p2 }

```

Fonte: Elaborada pelo autor.

A consulta para identificação da anomalia *Redundant implication* pode ser vista no Código-fonte 27. Ela identifica regras que possuem um átomo do consequente com um predicado que é uma superclasse de um predicado encontrado no antecedente da mesma regra.

Código-fonte 27 – Identificação da anomalia *Redundant implication* usando SPARQL

```

1: SELECT DISTINCT ?rule
2: WHERE { ?rule rdf:type metaFor:Rule ;
3:         metaFor:hasAntecedentAtom ?laa ;
4:         metaFor:hasConsequentAtom ?lca .
5:         ?laa metaFor:hasPredicate ?p1 ;
6:         metaFor:hasFirstArgument ?var1 .
7:         ?lca metaFor:hasPredicate ?p2 ;
8:         metaFor:hasFirstArgument ?var1 .
9:         ?p2 metaFor:hasSub ?p1 .
10:        ?p1 rdf:type metaFor:Class .
11:        ?p2 rdf:type metaFor:Class }

```

Fonte: Elaborada pelo autor.

A identificação da anomalia *Redundant implication of transitivity or symmetry* pode ser dividida em duas consultas, uma com o uso redundante de transitividade e outra com o uso redundante de simetria. Em ambos os casos, a ocorrência somente envolve átomos do antecedente das regras. No Código-fonte 28, as anomalias identificadas são regras que possuem o uso redundante de propriedades de objeto transitivas. No Código-fonte 29, as anomalias identificadas são regras que usam de forma redundante propriedades de objeto simétricas.

Código-fonte 28 – Identificação da anomalia *Redundant implication of transitivity* com propriedades transitivas usando SPARQL

```

1: SELECT DISTINCT ?rule
2: WHERE { ?rule metaFor:hasAntecedentAtom ?laa1 ;
3:         metaFor:hasAntecedentAtom ?laa2 ;
4:         metaFor:hasConsequentAtom ?lca .
5:         ?laa1 metaFor:hasPredicate ?p1 ;
6:         metaFor:hasFirstArgument ?arg1 ;
7:         metaFor:hasSecondArgument ?arg2 .
8:         ?laa2 metaFor:hasPredicate ?p2 ;
9:         metaFor:hasFirstArgument ?arg2 ;
10:        metaFor:hasSecondArgument ?arg3 .
11:        ?lca metaFor:hasPredicate ?p3 ;
12:        metaFor:hasFirstArgument ?arg1 ;
13:        metaFor:hasSecondArgument ?arg3 .
14:        ?p1 rdf:type metaFor:TransitiveObjectProperty;
15:        metaFor:isEquivalent ?p2 ;
16:        metaFor:isEquivalent ?p3 }

```

Fonte: Elaborada pelo autor.

Código-fonte 29 – Identificação da anomalia *Redundant implication of symmetry* com propriedades simétricas usando SPARQL

```

1: SELECT DISTINCT ?rule
2: WHERE { ?rule metaFor:hasAntecedentAtom ?laa ;
3:           metaFor:hasConsequentAtom ?lca .
4:         ?laa metaFor:hasPredicate ?p1 ;
5:           metaFor:hasFirstArgument ?arg1 ;
6:           metaFor:hasSecondArgument ?arg2 .
7:         ?lca metaFor:hasPredicate ?p2 ;
8:           metaFor:hasFirstArgument ?arg2 ;
9:           metaFor:hasSecondArgument ?arg1 .
10:        ?p1 rdf:type metaFor:SymmetricObjectProperty;
11:        metaFor:isEquivalent ?p2 }
```

Fonte: Elaborada pelo autor.

Assim como anomalia anterior, a anomalia *Redundant use of transitivity and symmetry* pode ser dividida em duas: transitividade e simetria. Nesse caso, o uso redundante de transitividade ou de simetria ocorre em uma combinação de átomos do antecedente com átomos do conseqüente. Nessa anomalia, o átomo redundante do conseqüente da regra gera uma afirmação que já é deduzida pela própria definição da propriedade transitiva ou simétrica. Para a identificação dessa anomalia com transitividade, o Código-fonte 30 contém a consulta SPARQL. Para simetria, o Código-fonte 31 apresenta a consulta SPARQL.

Código-fonte 30 – Identificação da anomalia *Redundant use of transitivity* com propriedades transitivas usando SPARQL

```

1: SELECT DISTINCT ?rule
2: WHERE { ?rule metaFor:hasAntecedentAtom ?laa1 ;
3:           metaFor:hasAntecedentAtom ?laa2 ;
4:           metaFor:hasAntecedentAtom ?laa3 .
5:         ?laa1 metaFor:hasPredicate ?p1 ;
6:           metaFor:hasFirstArgument ?arg1 ;
7:           metaFor:hasSecondArgument ?arg2 .
8:         ?laa2 metaFor:hasPredicate ?p2 ;
9:           metaFor:hasFirstArgument ?arg2 ;
10:          metaFor:hasSecondArgument ?arg3 .
11:         ?laa3 metaFor:hasPredicate ?p3 ;
```

```

12:          metaFor:hasFirstArgument ?arg1 ;
13:          metaFor:hasSecondArgument ?arg3 .
14:    ?p1 rdf:type metaFor:TransitiveObjectProperty;
15:          metaFor:isEquivalent ?p2 ;
16:          metaFor:isEquivalent ?p3 }

```

Fonte: Elaborada pelo autor.

Código-fonte 31 – Identificação da anomalia *Redundant use of symmetry* com propriedades simétricas usando SPARQL

```

1: SELECT DISTINCT ?rule
2: WHERE { ?rule metaFor:hasAntecedentAtom ?laa ;
3:          metaFor:hasConsequentAtom ?lca .
4:    ?laa metaFor:hasPredicate ?p1 ;
5:          metaFor:hasFirstArgument ?arg1 ;
6:          metaFor:hasSecondArgument ?arg2 .
7:    ?lca metaFor:hasPredicate ?p2 ;
8:          metaFor:hasFirstArgument ?arg2 ;
9:          metaFor:hasSecondArgument ?arg1 .
10:   ?p1 rdf:type metaFor:SymmetricObjectProperty;
11:         metaFor:isEquivalent ?p2 }

```

Fonte: Elaborada pelo autor.

Para realizar a identificação da anomalia *Self-contradicting rule*, foi decidido separar a consulta SPARQL em duas, de modo a simplificar o entendimento da identificação dessa anomalia. Para identificar regras SWRL em que a contradição na própria regra ocorre por meio de classes disjuntas foi elaborado o Código-fonte 32. Para contradição em regras SWRL, por meio de propriedades de objeto disjuntas, foi elaborado o Código-fonte 33.

Código-fonte 32 – Identificação da anomalia *Self-contradicting rule* em classes usando SPARQL

```

1: SELECT DISTINCT ?rule
2: WHERE { ?rule metaFor:hasAntecedentAtom ?a1 ;
3:          metaFor:hasConsequentAtom ?a2 .
4:    ?a1 metaFor:hasPredicate ?p1 ;
5:          metaFor:hasFirstArgument ?var1 .
6:    ?a2 metaFor:hasPredicate ?p2 ;
7:          metaFor:hasFirstArgument ?var1 .
8:    ?p1 metaFor:isDisjoint ?p2 .

```

```

9:          ?p1 rdf:type metaFor:Class .
10:         ?p2 rdf:type metaFor:Class }

```

Fonte: Elaborada pelo autor.

Código-fonte 33 – Identificação da anomalia *Self-contradicting rule* em propriedades de objetos usando SPARQL

```

1: SELECT DISTINCT ?rule
2: WHERE { ?rule metaFor:hasAntecedentAtom ?a1 ;
3:         metaFor:hasConsequentAtom ?a2 .
4:         ?a1 metaFor:hasPredicate ?p1 ;
5:         metaFor:hasFirstArgument ?var1 ;
6:         metaFor:hasSecondArgument ?var2 .
7:         ?a2 metaFor:hasPredicate ?p2 ;
8:         metaFor:hasFirstArgument ?var1 ;
9:         metaFor:hasSecondArgument ?var2 .
10:        ?p1 metaFor:isDisjoint ?p2 }

```

Fonte: Elaborada pelo autor.

A consulta para identificação da anomalia *Rule subsumption* pode ser vista no Código-fonte 34. Ela identifica conjuntos de duas regras em que o antecedente de uma regra subsume o antecedente de outra e os seus consequentes são iguais.

Código-fonte 34 – Identificação da anomalia *Rule subsumption* usando SPARQL

```

1: SELECT ?rule1 ?rule2
2: WHERE { ?rule1 metaFor:subsumes ?rule2 ;
3:         metaFor:sameConsequent ?rule2 }

```

Fonte: Elaborada pelo autor.

Além dessas identificações, durante o doutorado foram elaboradas algumas identificações de anomalias usando regras SWRL (no lugar de consultas SPARQL). Entretanto, por serem similares às identificações apresentadas em SPARQL, elas não são apresentadas neste trabalho podendo ser encontradas no artigo “*User Extensible System to Identify Problems in OWL Ontologies and SWRL Rules*” (ORLANDO; MUSEN; MOREIRA, 2015).

5.4 Visualização das anomalias

Como mostrado previamente na Figura 18, o sistema ONTO-Analyst possui um painel de visualização das anomalias identificadas. O painel de visualização foi desenvolvido utilizando um navegador Web embutido chamado WebView da API JavaFX². Ele foi utilizado para que fosse possível utilizar a biblioteca gráfica D3 (em JavaScript) e para facilitar uma migração futura do protótipo ONTO-Analyst para a web. A D3 é uma biblioteca para visualização de dados que usa as linguagens HTML, SVG, e CSS para gerar visualizações (BOSTOCK; OGIEVETSKY; HEER, 2011).

O sistema ONTO-Analyst mostra uma visualização em grafo em que as instâncias da ontologia MetaFOR são os nós e os relacionamentos entre elas são as arestas. Na Figura 20 é apresentado o painel com exemplo de visualização. O grafo mostrado possui três entidades envolvidas diretamente (borda vermelha) e 2 duas indiretamente (borda preta) na anomalia *Partition error in taxonomy*. Essa visualização permite mostrar não só as entidades diretamente envolvidas na anomalia mas também as indiretamente envolvidas, o que é uma grande vantagem. Ao analisar a visualização, o desenvolvedor pode perceber que o problema ocasionado pode estar relacionado com uma entidade envolvida indiretamente.

Figura 20 – Representação gráfica de uma ocorrência de anomalia onde as instâncias MetaFOR são nós e as arestas são relações entre as instâncias.

Fonte: Elaborada pelo autor.

Existem diversos tipos de nós utilizados em uma visualização no ONTO-Analyst (Figura 21). De um modo geral, os nós são representados como retângulos com símbolos no canto superior esquerdo que representam o tipo de entidade visualizada. Por exemplo, um círculo laranja representa uma classe. Alguns símbolos são semelhantes aos símbolos utilizados na ferramenta Protégé, como os utilizados para classes e propriedades. Os símbolos do Protégé foram utilizados devido à popularidade dessa ferramenta (KHONDOKER; MUELLER, 2010). Outros símbolos, como os para regras e átomos, que não existem na

² JavaFX: <https://docs.oracle.com/javafx/2/api/>

ferramenta Protégé, foram criados especialmente para o protótipo ONTO-Analyst. Eles podem ser vistos na Figura 21.

Figura 21 – Tipos de nós disponíveis para a visualização das ontologias no sistema ONTO-Analyst. Cada nó é um retângulo com um símbolo para representar um tipo de entidade analisada.

Fonte: Elaborada pelo autor.

Além de símbolos para nós, uma representação inicial para arestas foi criada. Arestas são linhas que conectam nós para representar o relacionamento entre as entidades visualizadas. Elas tem símbolos representando o tipo de relacionamento envolvido entre os nós (Figura 22).

Figura 22 – Tipos de arestas para a visualização das ontologias no sistema ONTO-Analyst. Cada aresta é uma linha com um símbolo que representa o tipo de relacionamento entre os nós

Fonte: Elaborada pelo autor.

5.4.1 Criação dos grafos

O grafo, mostrado na Figura 20, utiliza os nós e arestas descritos anteriormente. Esse grafo mostra duas classes disjuntas que têm uma mesma subclasse na hierarquia. Para construí-lo, o sistema lê o resultado da identificação da anomalia, selecionado pelo usuário, e a consulta SPARQL que o gerou. As entidades, selecionadas no resultado da identificação, se tornam os nós principais do grafo e a eles, quando existir, são adicionados seus nós intermediários referentes a hierarquia de classes/propriedades. A partir desses nós, são definidas as suas arestas com base nas definições dos filtros encontrados na cláusula *where* da consulta SPARQL. Por último, o sistema ONTO-Analyst define a posição dos nós na visualização, baseado no requisito de que a hierarquia deve ser representada na vertical, e plota o grafo utilizando a biblioteca D3.

5.5 Considerações finais

Neste Capítulo foi apresentado o método desenvolvido ONTO-Analyst. De um modo geral, esse método é composto por uma ontologia sobre ontologias (MetaFOR), uma versão da ontologia a ser analisada no formato MetaFOR e um conjunto de consultas SPARQL que seleciona anomalias. Essas anomalias podem ser visualizadas usando grafos. As ontologias no formato MetaFOR possuem informações estruturais extraídas das ontologias analisadas. Essa representação permite que sejam utilizadas consultas SPARQL para identificação de anomalias.

O protótipo do sistema ONTO-Analyst implementa o método ONTO-Analyst. Ele extrai as informações das ontologias analisadas e as converte para o formato MetaFOR. Ele executa as consultas SPARQL para identificar as anomalias. Por último, ele visualiza as anomalias encontradas.

No próximo capítulo, são apresentados dois tipos de experimentos de validações, realizados com o ONTO-Analyst e a MetaFOR, e seus resultados. No primeiro tipo, são realizados experimentos quantitativos na identificação de anomalias em ontologias de repositórios públicos importantes. No segundo tipo, é apresentado um estudo de caso com visualizações de anomalias encontradas em ontologias públicas.

EXPERIMENTOS

As duas partes principais do método ONTO-Analyst são a identificação e a visualização de anomalias em ontologias. Para cada uma delas foram elaborados experimentos para validar e mostrar a importância das duas partes para desenvolvedores de ontologias.

A validação da identificação de anomalias é apresentada na Seção 6.1, por meio de uma análise quantitativa dos resultados de experimentos da identificação de anomalias, realizados em um grande conjunto de ontologias públicas. Já a validação da visualização de anomalias identificadas é apresentada na Seção 6.2, por meio de um estudo de caso sobre visualizações geradas de anomalias encontradas em ontologias públicas.

6.1 Análise quantitativa da ocorrência de anomalias em ontologias públicas

Esta análise consiste em executar a identificação de anomalias em ontologias encontradas em repositórios públicos. Para tanto, foram selecionadas 18 anomalias representativas, extraídas da literatura científica (Seção 3.4) e implementadas suas identificações em SPARQL (Seção 5.3). Para executar as identificações de anomalias, um conjunto representativo de ontologias foi selecionado. Ao total foram selecionadas 608 ontologias de 4 repositórios públicos e 2 artigos, segundo os critérios que serão discutidos da Seção 6.1.1.

O objetivo principal dos experimentos, relacionados à detecção de anomalias, foi demonstrar que o sistema ONTO-Analyst consegue identificar um conjunto representativo de anomalias num conjunto representativo de ontologias. Além desse objetivo principal, a análise dos resultados serviu para encontrar os tipos de anomalias mais comuns encontrados em ontologias.

Para alcançar esses objetivos, os experimentos foram divididos em 4 etapas:

1. Seleção dos repositórios públicos de ontologias (Subseção 6.1);
2. Download das ontologias escolhidas (Subseção 6.1.2);
3. Conversão das ontologias escolhidas para o formato MetaFOR (Subseção 6.1.3);
4. Identificação e análise das anomalias implementadas (Subseção 6.1.4).

6.1.1 Seleção dos repositórios de ontologias

Para escolher os repositórios de ontologias, foi elaborada uma lista de repositórios recomendados para publicação de ontologias, com base na recomendação de d'Aquin e Noy (2012) e do grupo de pesquisa em OWL da Universidade de Manchester¹. Para todos os 18 repositórios públicos das duas listas, foram aplicados os seguintes critérios de seleção:

- Ter ontologias disponíveis em formatos suportados pela OWL API;
- Ser possível o download de todas as ontologias sem necessidade de intervenção humana.

Com esses critérios, foram selecionados os seguintes repositórios de ontologias:

Dumontier Lab Repository - É um repositório para bio-ontologias que representam conceitos e relações biológicas e científicas. Esse repositório será subsequentemente referido nos experimentos como REP1.

TONES - É um serviço web com ontologias utilizadas principalmente para testes. Ele contém uma variedade de diferentes ontologias, variando em expressividade e tamanho. Esse repositório será subsequentemente referido como REP2.

BioPortal - É um repositório para ontologias biomédicas que foi desenvolvido pelo *National Center for Biomedical Ontology* (NCBO). Ele permite que pesquisadores avaliem ontologias e contribuam para o desenvolvimento de conteúdo das mesmas. Esse repositório será subsequentemente referido como REP3.

Além das ontologias desses repositórios, duas ontologias com regras em SWRL foram adicionadas a partir de artigos. Elas serão representadas como pelo acrônimo REP-PAPER. No total, esses repositórios somam 546 ontologias, porém apenas 8 ontologias contém regras SWRL, com um total de 452 regras SWRL nelas.

Para resolver essa falta de ontologias com regras para os experimentos quantitativos, outros repositórios de ontologias (com ontologias que contivessem regras SWRL) foram buscados, sem sucesso. Por essa razão, foram procuradas ontologias com regras

¹ Lista de repositórios de ontologias: <http://owl.cs.manchester.ac.uk/tools/repositories/>

SWRL no repositório GitHub². O GitHub é um conhecido repositório de códigos-fontes que tem milhares de projetos. Nesse repositório foram encontradas diversas ontologias com regras SWRL, sendo assim, ele também foi adicionado aos experimentos.

Como o GitHub não é um repositório específico de ontologias, foi criado um critério de seleção específico para as ontologias encontradas nele: as ontologias deveriam ser escritas em OWL, conter regras em SWRL e passar pelas verificações dos *reasoners* Pellet (versão 2.3.1) e Hermit (versão 1.3.8) sem apresentar nenhum erro. Esse critério foi usado para excluir ontologias com erros deliberados, pois haviam ontologias criadas para serem usadas em testes de programas, como o *reasoner* Pellet (projeto encontrado no GitHub). No total, foram utilizadas 82 ontologias com regras SWRL obtidas no GitHub. Esse grupo de ontologias será subsequentemente referido como REPGIT.

6.1.2 Download das ontologias

Os locais em que as ontologias foram encontradas são os seguintes:

- REP1: dumontierlab.com/index.php?page=ontologies
- REP2: rpc295.cs.man.ac.uk:8080/repository/browser³
- REP3: bioportal.bioontology.org/ontologies
- REPPAPER: Dos artigos de [Hastings et al. \(2010\)](#) e [Young et al. \(2009\)](#).
- REPGIT: github.com/search?q=extension%3Aowl+swrl+imp

Para o download das ontologias dos repositórios REP1, REP2 e REP3, foi criada uma ferramenta que procura todas as URLs de ontologias nos repositórios e realiza o download automático dessas ontologias. Algumas ontologias estão disponíveis em mais de um arquivo com diferentes extensões. Nesse caso, foi dada prioridade à seguinte ordem de extensões de arquivos: “.owl”, “.xrd”, “.ttl” e “.obo”. A Tabela 4 mostra que, em abril de 2015, foi realizado o download de 71 ontologias do REP1, 219 do REP2 e 254 do REP3. No REPGIT, o download das ontologias foi realizado manualmente em 7 e 8 de maio de 2015. No total, foram obtidas 628 ontologias para os experimentos.

6.1.3 Conversão para o formato MetaFOR

Cada uma das 628 ontologias baixadas foi submetida à conversão para o formato MetaFOR pelo sistema ONTO-Analyst. Os resultados gerais desse processo de conversão

² GitHub: <https://github.com/>

³ Este serviço web está atualmente descontinuado. As ontologias estão armazenadas em <https://zenodo.org/record/32717>

Tabela 4 – Repositórios com o número de ontologias e suas datas de download.

Nome	Data de download	Número de ontologias
REP1	13 de abril de 2015	71
REP2	13 de abril de 2015	219
REP3	26 de abril de 2015	254
REPPAPER		2
REPGIT	7 e 8 de maio de 2015	82

Fonte: Elaborada pelo autor.

podem ser vistos na Tabela 5. Para demonstrar a qualidade das ontologias analisadas, todas as ontologias foram verificadas com o *reasoner* Pellet (versão 2.3.1) ou Hermit (versão 1.3.8) antes da conversão. A utilização dos dois *reasoners* foi necessária porque algumas ontologias podem: (i) ter afirmações que o *reasoner* Hermit não suporta (SWRL built-ins) ou (ii) ter um desempenho muito lento do *reasoner* Pellet. O Hermit foi usado primeiro e, em caso de insucesso do mesmo, o Pellet foi usado na sequência. Esse processo serviu para mostrar a qualidade das ontologias analisadas e não para excluir ontologias da análise.

Tabela 5 – Resultados gerais do processo de conversão das ontologias analisadas para o formato MetaFOR

Nome do repositório	Número de ontologias	Ontologias convertidas e validadas por <i>reasoners</i>	Ontologias convertidas e não validadas por <i>reasoners</i>	Não convertidas
REP1	71	69	0	2
REP2	219	199	14	6
REP3	254	225	17	12
REPPAPER	2	2	0	0
REPGIT	82	82	–	–
Totais	628	577	31	20

Fonte: Elaborada pelo autor.

Após a realização da conversão das ontologias, se obteve os resultados gerais presentes na Tabela 5. Nessa tabela, pode ser visto que 577 ontologias foram validadas por um *reasoner* e 31 não foram, mas todas as 608 ontologias foram convertidas para o formato MetaFOR sem apresentar problemas. As ontologias não validadas apresentaram problemas de inconsistências, atingiram o tempo máximo alocado para validação (24 horas) ou tiveram problemas de falta de memória RAM. Ainda na Tabela 5, pode ser visto que, no total, 608 ontologias foram convertidas para o formato MetaFOR, sendo que as 20 restantes não foram convertidas por dificuldades da OWL API em ler seus arquivos. A eficácia do processo de conversão foi de 96,8%.

As 31 ontologias convertidas que não passaram pela validação dos *reasoners* (coluna 4, Tabela 5) são desmembradas em três categorias de problemas enfrentados pelos *reasoners* (Tabela 6). Esses problemas foram: inconsistências (16 ontologias), *reasoner* rodando por mais de 24 horas (11 ontologias) e falta de memória durante a validação (4 ontologias). Cada um desses problemas é mostrado numa coluna da Tabela 6 com o total de ontologias para cada repositório. As especificações do computador utilizado para as validações/conversões são:

- Intel® Core™ i7-990X com 6 núcleos.
- 16 GB de memória RAM.
- Windows 8.

Tabela 6 – Ontologias convertidas para o formato MetaFOR que falharam na validação pelos *reasoners*.

Nome do repositório	Inconsistências encontradas pelo <i>reasoner</i>	<i>Reasoner</i> rodando > 24 horas	Falta de memória
REP2	7	7	0
REP3	9	4	4
Totais	16	11	4

Fonte: Elaborada pelo autor.

As 20 ontologias que não foram convertidas, apresentadas anteriormente na Tabela 5, podem ser classificadas em dois tipos de problemas: ontologias com *imports* que não existem mais (8) e ontologias em que a OWL API não conseguiu ler o formato dos seus arquivos (12). Esses dois tipos de problemas são apresentados na Tabela 7. Os dois problemas ocorridos não são problemas relacionados diretamente com o ONTO-Analyst, sendo assim pode-se supor que o mesmo seria capaz de convertê-las, caso a OWL API pudesse carregá-las.

6.1.3.1 Resumo das estruturas convertidas

A sumarização, mostrada na Tabela 8, apresenta a vasta quantidade de dados que foram analisados nos experimentos.

As ontologias analisadas têm muitas classes, em média ~7.199 por ontologia. A média para propriedades de objetos foi de ~38 e de propriedades de dados foi ~14 por ontologia. Todas as ontologias contêm pelo menos uma propriedade de objeto e uma classe. 979 regras existem em 90 ontologias. Surpreendentemente, foi encontrado um número muito pequeno de definições de cardinalidades. Cardinalidades deveriam ser importantes

Tabela 7 – Ontologias descartas com problemas no processo de conversão para o formato MetaFOR

Nome do repositório	Problemas de importação	Problemas na leitura do formato
REP1	1	1
REP2	6	0
REP3	1	11
Totais	8	12

Fonte: Elaborada pelo autor.

Tabela 8 – Resumo das estruturas nas 608 ontologias analisadas

	Classes	Propriedades de objetos	Propriedades de dados	Regras	Cardinalidades
Soma	4.377.570	23.593	8.487	979	7
Número de ontologias com esta entidade	608	608	599	90	4
Média por ontologia	~7.199	~38	~14	~1,6	~0,01
Máximo por ontologia	403.295	950	1.380	159	4

Fonte: Elaborada pelo autor.

restrições em ontologias, mas, nas 608 ontologias analisadas, apenas 7 dessas restrições foram encontradas.

6.1.4 Identificação e análise das anomalias

Após escolher as anomalias a serem identificadas (Seção 3.4) e escrever as consultas em SPARQL para encontrá-las (Seção 5.3), o sistema ONTO-Analyst foi utilizado para encontrar tais anomalias nas 608 ontologias convertidas para o formato MetaFOR. Esta subseção apresenta os resultados da identificação dessas anomalias. Uma visão geral dos resultados é apresentada na Seção 6.1.4.1, seguida da Seção 6.1.4.2, com uma visão detalhada de cada anomalia detectada nas ontologias. Finalmente, os resultados de algumas ontologias mais importantes são apresentados na Seção 6.1.4.3.

6.1.4.1 Visão geral dos resultados

Os resultados gerais são agrupados pela principal entidade envolvida na anomalia, podendo ser classes, propriedades, regras ou restrições de cardinalidade. Nas próximas três tabelas, os resultados das anomalias que ocorrem em classes, propriedades e regras

são mostrados. Os resultados para cardinalidades não são apresentados porque o sistema ONTO-Analyst não encontrou nenhuma ocorrência de anomalia nessas restrições.

Os resultados de anomalias que ocorrem em classes podem ser vistos na Tabela 9. Essa tabela mostra sete anomalias que podem ocorrer em classes. As anomalias *Lazy class or property*, *Redundancy by repetitive taxonomic definition* e *Exact circularity in taxonomy* ocorrem tanto em propriedades quanto em classes, mas são apresentadas apenas nessa tabela.

Tabela 9 – Resultados gerais das identificações de anomalias que ocorrem em classes/propriedades

Anomalias	Ocorrências totais	Ontologias com anomalias	Máximo de ocorrências em uma ontologia
<i>Exact circularity in taxonomy</i>	583	70	79
<i>Partition error in taxonomy</i>	3.289	52	2.028
<i>Redundancy by repetitive taxonomic definition</i>	45.894	265	10.825
<i>Lonely disjoint classes</i>	4.325	116	932
<i>Chains of inheritance</i>	811.726	487	106.293
<i>Lazy class or property</i>	3.556.207	592	375.836

Fonte: Elaborada pelo autor.

A primeira linha na Tabela 9, referente à *Exact circularity in taxonomy*, é uma anomalia que ocorreu 583 vezes em classes. Essa circularidade foi a que menos ocorreu nesse grupo de anomalias. Deve-se lembrar que mais de 4 milhões de classes foram analisadas. Essa anomalia ocorreu em apenas 70 ontologias, sendo que uma ontologia teve 79 ocorrências.

A segunda linha na mesma tabela, referente à anomalia *Partition error in taxonomy*, mostra uma inconsistência que ocorreu 3.289 vezes. É importante enfatizar que mais de 61% destas ocorrências (2.028 ocorrências) são encontradas em apenas uma ontologia. Na próxima subseção, essa situação será discutida em mais detalhes. Essa anomalia foi a que ocorreu em menos ontologias analisadas desse grupo. Apenas 52 ontologias apresentaram pelo menos uma ocorrência de *Partition error in taxonomy*. Isto se deve ao fato de que os *reasoners* conseguem mais facilmente identificar inconsistências.

A terceira linha, referente à anomalia *Redundancy by repetitive taxonomic definition*, é uma redundância na definição de subclasses ou subpropriedades que ocorre em ~1% de todas as classes e propriedades analisadas. Desse total, 1/4 ocorre em apenas uma ontologia.

As outras três linhas são referentes a três deficiências (anomalias no design da ontologia). A anomalia *Lonely disjoint classes* foi encontrada mais de 4 mil vezes. A ano-

malia *Chains of inheritance* foi encontrada mais de 800 mil vezes. Por último, a anomalia *Lazy class or property* foi encontrada mais de 3 milhões de vezes e foi a anomalia que mais ocorreu entre todos os tipos de entidades.

A Tabela 10 mostra os resultados para anomalias que ocorrem apenas em propriedades de objetos. A única anomalia que ocorre apenas em propriedades de objetos é a *Circular properties* (tipo de circularidade). No total, mais de 23 mil propriedades de objetos foram analisadas e apenas 28 continham essa anomalia. Essa anomalia ocorreu em apenas 5 ontologias.

Tabela 10 – Resultados gerais das identificações de anomalias que ocorrem em propriedades de objetos

Anomalias	Ocorrências totais	Ontologias com anomalias	Máximo de ocorrências em uma ontologia
<i>Circular properties</i>	28	5	10

Fonte: Elaborada pelo autor.

A Tabela 11 apresenta as anomalias que ocorreram nas 979 regras analisadas. A primeira anomalia analisada é um tipo de circularidade, as três seguintes são inconsistências e o restante são redundâncias. A primeira anomalia (*Circularity between rules and taxonomy*) foi encontrada 114 vezes em 19 ontologias e foi a anomalia mais comum em regras. Nenhuma anomalia foi encontrada nas três inconsistências analisadas, o que pode ser devido ao fato de os *reasoners* encontrarem mais facilmente esse tipo de anomalia. Quanto às anomalias de redundâncias, a anomalia *Redundant implication* foi a única anomalia a não ser encontrada nas ontologias analisadas. É importante ressaltar que o número de ontologias com regras SWRL analisado foi limitado.

A influência dos repositórios de ontologia nos experimentos é apresentada em dois gráficos. A Figura 23 mostra a influência de cada repositório na ocorrência de cada tipo de anomalia. Cada barra corresponde a 100% da ocorrência de cada tipo de anomalia e cada cor nas barras representa um repositório. Nesse gráfico, é possível analisar que o REP3 foi o repositório que teve mais influência em ocorrências de inconsistências, redundâncias e deficiências. O REP2 foi o repositório que teve mais influência em anomalias do tipo circularidade, praticamente não tendo muita influência em mais nenhum tipo de anomalia. O REPGIT teve alguma influência em anomalias do tipo circularidade. Os repositórios REP1 e REPPAPER não tiveram influência significativa nas anomalias encontrados nesses experimentos.

A Figura 24 mostra a influência da ocorrência de anomalias por tipo de entidade de cada repositório. Cada barra corresponde a 100% das ocorrências de anomalias para um tipo de entidade. Cada cor nas barras representa um repositório. A figura mostra que

Tabela 11 – Resultados gerais das identificações de anomalias que ocorrem em regras

Anomalias	Ocorrências totais	Ontologias com anomalias	Máximo de ocorrências em uma ontologia
<i>Circularity between rules and taxonomy</i>	114	19	20
<i>Contradicting rules</i>	0	0	0
<i>Incompatible rule antecedent</i>	0	0	0
<i>Self-contradicting rule</i>	0	0	0
<i>Redundant derivation in the antecedent</i>	13	1	13
<i>Redundant implication</i>	0	0	0
<i>Redundant implication of transitivity or symmetry</i>	1	1	1
<i>Redundant use of transitivity and symmetry</i>	4	2	2
<i>Rule subsumption</i>	4	2	2

Fonte: Elaborada pelo autor.

quase todas as anomalias de classes ocorreram no repositório REP3 e o restante ocorreu no repositório REP2. Com relação as propriedades, a grande maioria das ocorrências de anomalias foram encontradas no repositório REP2, seguido por REP3 e REPGIT. Com relação as regras, a maior ocorrência de anomalias foi no repositório REPGIT (o que já era esperado), seguido por REP3 e REP2. Pode ser notado que, mesmo contendo apenas duas ontologias, o repositório REPPAPER mostrou alguma influência nos resultados da identificação de anomalias em regras.

6.1.4.2 Análise de algumas anomalias específicas

Nesta subseção, algumas situações pertinentes serão analisadas de forma a ilustrar as anomalias identificadas. Para cada anomalia foram feitas diversas análises para identificar ocorrências interessantes ou mostrar algum padrão geral de todo o grupo de ocorrências. Dentre todas essas análises, foram escolhidas as mais relevantes que serão

Figura 23 – Influência da origem do repositório nos tipos de anomalias.

Fonte: Elaborada pelo autor.

Figura 24 – Influência da ocorrência de anomalias por tipo de entidade de cada repositório.

Fonte: Elaborada pelo autor.

apresentadas em cada uma das próximas subseções.

6.1.4.2.1 Exact circularity in taxonomy

Com relação a essa circularidade, três tipos de ocorrências foram encontrados, sendo que cada um deles mostra uma situação diferente nas ontologias analisadas. A primeira situação ocorreu quando uma classe/propriedade tem a si mesma como sua sub-classe/subpropriedade. Essa situação foi encontrada, por exemplo, na ontologia *Software*

Ontology (SWO)⁴, no repositório REP3. Nessa ontologia, existe uma classe, chamada **ObsoleteClass**, que é uma subclasse dela mesma. Essa ontologia tem 4.068 classes e essa classe é a única ocorrência da circularidade *Exact circularity in taxonomy*. Provavelmente essa afirmação (axioma) foi gerada por um engano no desenvolvimento dessa ontologia.

A segunda situação é quando uma classe/propriedade tem a definição de equivalência com sua subclasse/subpropriedade. Por exemplo, a ontologia *Biological Imaging Methods Ontology* (FBbi)⁵, do repositório REP3, tem uma ocorrência dessa circularidade nas classes FBbi_00000419 e FBbi_00000013. FBbi_00000419 tem FBbi_00000013 como sua superclasse e sua classe equivalente. Essa definição provavelmente é um erro pois a definição de equivalência torna as duas classes iguais, não existindo mais a necessidade da relação de herança.

A terceira situação ocorre quando 4 entidades diferentes criam essa circularidade. Uma ontologia que relata terminologias e conceitos usados para nutrição humana em uma configuração clínica e biomédica, chamada *Bionutrition Ontology* (BNO)⁶, apresenta uma ocorrência dessa situação. As classes **FAT**, **Poly**, **Saturated** e **Unsaturated** formam uma circularidade em que cada uma das classes é uma subclasse das outras três. Se a classe X é subclasse de Y e Y é subclasse de X, então todas as instâncias de X são instâncias de Y e todas as instâncias de Y são instâncias de X. Dessa forma, X e Y são equivalentes. É possível concluir que as classes **FAT**, **Poly**, **Saturated** e **Unsaturated** são equivalentes o que cria uma circularidade nessa ontologia. Dados os nomes das classes, isto provavelmente não é o que os autores pretendiam.

Encontrar esse tipo de situação pode ser complicado em uma ontologia com muitas classes e propriedades, mas o sistema ONTO-Analyst permite aos usuários facilmente encontrar situações como essas. Todas essas diferentes identificações foram realizadas com a mesma consulta SPARQL. Essa é uma grande vantagem, pois permite aos usuários procurar por algo mais específico ou por algo mais genérico, como nas identificações mostradas nesta subseção.

6.1.4.2.2 Partition error in taxonomy

Essa foi a única anomalia do tipo inconsistência encontrada nas ontologias analisadas, sendo que foram encontradas ocorrências em 52 diferentes ontologias. Na ontologia *Neuroscience Information Framework Standard Ontology* (NIFSTD)⁷ essa inconsistência ocorreu 2.028 vezes. Analisando as ocorrências, as classes **Continuant** e **Occurrent** são responsáveis por quase todas as ocorrências encontradas nessa ontologia. Essas classes

⁴ Ontologia SWO: <http://biportal.bioontology.org/ontologies/SWO>

⁵ Ontologia FBbi: <http://biportal.bioontology.org/ontologies/FBbi>

⁶ Ontologia BNO: <http://biportal.bioontology.org/ontologies/BNO>

⁷ Ontologia NIFSTD: <http://biportal.bioontology.org/ontologies/NIFSTD>

são disjuntas e as diversas classes que são subclasses de ambas fazem parte dessa anomalia. Provavelmente, um problema deve ter ocorrido durante a construção dessa ontologia, tendo em vista que não faz sentido ter subclasses que ao mesmo tempo pertencem às classes disjuntas *Continuant* e *Occurrent*.

Outra ocorrência dessa anomalia ocorreu na mesma ontologia: a classe *BSP0_0000126* é disjunta dela mesma. Então, ela e suas duas subclasses não podem ter instâncias.

6.1.4.2.3 Redundancy by repetitive taxonomic definition

Analisando essa redundância foi descoberto que ontologias que contém a anomalia *Partition error in taxonomy* também contém *Redundancy by repetitive taxonomic definition*. Isso ocorreu devido ao fato de que os *reasoners* criaram muitas relações *rdfs:subClassOf* nas classes associadas com essa inconsistência. Não foi encontrado o possível motivo para isso, então foi preferido descartar a influência de *Partition error in taxonomy* nessa anomalia. Para eliminar essa influência da análise da anomalia *Redundancy by repetitive taxonomic definition*, as ontologias que continham a anomalia *Partition error in taxonomy* foram descartadas da análise feita nessa subseção. De um total de 265 ontologias, 46 foram descartadas. As ontologias descartadas continham 25.530 ocorrências dessa anomalia, de um total de 45.894. Com as ocorrências restantes, foi criado um gráfico de dispersão, mostrado na Figura 25.

Figura 25 – Relação da porcentagem de ocorrência da anomalia *Redundancy by repetitive taxonomic definition* com o total de classes e propriedades de cada ontologia (cada ponto representa uma ontologia).

Fonte: Elaborada pelo autor.

A Figura 25 apresenta a porcentagem de ocorrência dessa anomalia no eixo vertical, o total de classes e propriedades no eixo horizontal, e cada ponto representa uma

ontologia. A porcentagem de ocorrências dessa anomalia, em cada ontologia, foi calculada pela divisão do total das classes/propriedades que apresentam a anomalia pelo total de classes/propriedades. O gráfico de dispersão mostra que a porcentagem de ocorrência dessa anomalia é maior em ontologias menores.

6.1.4.2.4 Lonely disjoint class

Para mostrar essa deficiência, uma ontologia do repositório REP2 foi escolhida. O nome da ontologia é *Gene Regulation Ontology* (GRO)⁸ e uma parte da sua hierarquia de classes é mostrada na Figura 26. Nessa figura, a classe *Locus* está selecionada no lado esquerdo e as classes disjuntas de *Locus* são mostradas no lado direito.

Figura 26 – Exemplo de uma ocorrência da anomalia *Lonely disjoint class* na ontologia GRO.

Fonte: Elaborada pelo autor.

A classe selecionada é disjunta das classes **Conformation**, **Direction**, **Orientation** e **Phenotype**, todas do mesmo ramo da hierarquia de classes, e também é disjunta das classes **PackedDNA** e **UnpackedDNA**, de outro ramo. Esse é um exemplo de uma situação em que pode ser necessária a atenção dos autores da ontologia, pois pode gerar problemas no futuro. Em algumas situações, a ontologia pode ser redesenhada de forma a consertar estas áreas mal projetadas.

⁸ Ontologia GRO: <http://doi.org/10.5281/zenodo.32717>

6.1.4.2.5 Lazy class or property

Essa anomalia foi a mais encontrada dentre todos os tipos de anomalias. Para mostrar a distribuição dessa anomalia, as ontologias foram divididas em 5 grupos, de acordo com a porcentagem de suas classe/propriedades afetadas por ela. A Figura 27 mostra, em um gráfico de setores, o tamanho relativo dos 5 grupos. Lembre que, se uma ontologia é composta por apenas uma taxonomia, sem outras restrições ou instâncias, ela tem 100% de ocorrências dessa anomalia.

No gráfico de setores, o maior grupo é das ontologias com 60 a 80% de classes/-propriedades identificadas com essa anomalia. Esse grupo e o grupo de 80% a 100% representam mais da metade das ontologias analisadas. Isso mostra que a maioria das 608 ontologias analisadas são compostas principalmente por taxonomias puras sem instâncias, restrições ou regras ligadas a elas.

Figura 27 – Distribuição da porcentagem de ocorrência da anomalia *Lazy class or property* com o total de classes e propriedades de cada ontologia

Fonte: Elaborada pelo autor.

Na Figura 28, outra análise é apresentada com um gráfico de dispersão em que os pontos azuis representam cada ontologia e a linha pontilhada preta representa a tendência linear obtida entre a relação das ocorrências de *Lazy class or property* e o total de classes e propriedades. A linha de tendência linear é o melhor ajuste para os dados em questão. Linhas de tendência lineares mostram que algo está aumentando ou diminuindo em um ritmo linear. Esse gráfico mostra que a relação entre as ocorrências da anomalia *Lazy class or property* (eixo vertical) e o total de classes/propriedades em cada ontologia (eixo horizontal) é diretamente proporcional. Essa anomalia é completamente relacionada com a quantidade de classes e propriedades em cada ontologia.

Figura 28 – Relação entre a ocorrência da anomalia *Lazy class or property* e o total de classes/-propriedades em cada ontologia.

Fonte: Elaborada pelo autor.

6.1.4.2.6 Circular properties

De forma a mostrar essa circularidade, foi escolhida uma ontologia do repositório REP3 chamada *International Classification of Functioning, Disability and Health* (ICF)⁹. Nessa ontologia, seis propriedades de objetos com essa circularidade foram encontradas.

A Figura 29 mostra a ocorrência dessa anomalia em duas propriedades de objeto que são inversas mas tem o mesmo `rdfs:domain` e o mesmo `rdfs:range: primaryCareParent` (na parte esquerda da figura) e `primaryCareSubClasses` (na parte direita da figura). A consequência dessa anomalia é que todas as instâncias que tem uma dessas propriedades serão classificadas tanto na classe `LinearizationSection` quanto na classe `ICDSection`.

Figura 29 – Ocorrência da anomalia *Circular properties* na ontologia ICF. No lado esquerdo está a definição da propriedade `primaryCareParent`. No lado direito está a definição da propriedade `primaryCareSubclasses`. Cada propriedade é inversa da outra, mas ambas tem o mesmo `rdfs:domain` e `rdfs:range`.

Fonte: Elaborada pelo autor.

⁹ Ontologia ICF: <http://bioportal.bioontology.org/ontologies/ICF>

6.1.4.2.7 Circularity between rules and taxonomy

Um exemplo interessante dessa circularidade foi encontrado na ontologia *Breast Cancer Grading Ontology* (BCGO)¹⁰ do repositório REP3. Ela é usada para atribuir uma pontuação para determinado tumor de câncer de mama.

Nessa ontologia, é encontrada uma regra (Código-fonte 35) que cria uma circularidade entre a classe `Nucleus` e sua subclasse `Mitosis`. Essa definição deveria ser considerada como uma relação restrita de `rdfs:subClassOf` entre `Nucleus` e `Mitosis`.

Código-fonte 35 – Regra da ontologia BCGO com a anomalia *Circularity between rules and taxonomy*

```
1: Nucleus(?x), hasEccentricity(?x, ?value),
2: greaterThan(?value, "0"^^int), lessThan(?value, "1"^^int)
3: ->
4: Mitosis(?x)
```

Fonte: Elaborada pelo autor.

6.1.4.2.8 Redundant derivation in the antecedent

Essa anomalia foi encontrada 13 vezes em uma ontologia do repositório REPGIT, chamada *ContextOntology*¹¹. Uma ocorrência dessa redundância em regras SWRL pode ser vista no Código-fonte 36.

Código-fonte 36 – Regra da ontologia contextOntology com a anomalia *Redundant derivation in the antecedent*

```
1: alarm-state-sensor(?alarmRef), context-element(?alarmRef),
2: context-element(?faceRef), face-recognition-sensor(?faceRef),
3: has-value-of-service(?alarmRef, ?alarmValue),
4: has-value-of-service(?faceRef, ?faceValue),
5: equal(?alarmValue, "0"^^int), equal(?faceValue, "1"^^int)
6: ->
7: AcceptableSensorValue(AlarmStateSensorI, true)
```

Fonte: Elaborada pelo autor.

A redundância nessa regra ocorre porque a classe `face-recognition-sensor` é uma subclasse de `context-element`. Em outras palavras, o átomo `context-element(?faceRef)`

¹⁰ Ontologia BCGO: <http://bioportal.bioontology.org/ontologies/BCGO>

¹¹ Ontologia *ContextOntology*: <https://goo.gl/5NidDu>

não é necessário na regra porque todas as instâncias de `face-recognition-sensor` são instâncias de `context-element`.

6.1.4.2.9 Redundant implication of transitivity or symmetry

Redundant implication of transitivity or symmetry e *Redundant use of transitivity and symmetry* são situações similares. Logo, apenas o caso da anomalia *Redundant implication of transitivity anomaly* é mostrado. O sistema ONTO-Analyst identificou uma regra, descrita no Código-fonte 37, da ontologia *Rules for processing graph knowledge-base*¹² do repositório REP1. Essa ontologia usa regras para processamento de uma base de conhecimentos gráficos.

Código-fonte 37 – Regra com a anomalia *Redundant implication of transitivity or symmetry*

```
1: isBefore(?graph-example-1.owlx, ?graph-example-1.owlly),
2: isBefore(?graph-example-1.owlly, ?graph-example-1.owlz),
3: DifferentFrom (?graph-example-1.owlx, ?graph-example-1.owlly),
4: DifferentFrom (?graph-example-1.owlx, ?graph-example-1.owlz),
5: DifferentFrom (?graph-example-1.owlz, ?graph-example-1.owlly)
6: ->
7: isBefore(?graph-example-1.owlx, ?graph-example-1.owlz),
8: select(?graph-example-1.owlx, ?graph-example-1.owlz)
```

Fonte: Elaborada pelo autor.

Essa regra usa a propriedade de objeto transitiva `isBefore` três vezes, duas vezes no antecedente e uma no consequente. Um *reasoner* pode deduzir este átomo consequente sem essa regra. Em outras palavras, com as definições $P(x, y)$ e $P(y, z)$, *reasoners* podem concluir $P(x, z)$ sem essa regra. Então o átomo `isBefore(?graph-example-1.owlx, ?graph-example-1.owlz)` do consequente não é necessário.

6.1.4.2.10 Rule subsumption

Essa anomalia ocorreu em duas ontologias. A ocorrência mais interessante ocorreu na ontologia de [Young et al. \(2009\)](#) do repositório REPPAPER. Ela foi desenvolvida para que pesquisadores pudessem compartilhar e integrar dados sobre autismo ([YOUNG et al., 2009](#)). As duas regras onde essa anomalia ocorre são apresentadas no Código-fonte 38.

Código-fonte 38 – Regra com a anomalia *Rule subsumption*

¹² Ontologia *Rules for processing graph knowledgebase*: <http://ontology.dumontierlab.com/graph-rules.owl>

```

1: AUTISMC000000(?a),
2: ADI_2003_acqorlossoflang_aphrase(?a, ?phraseage),
3: SubjectKey(?a, ?subjectID),
4: createOWLThing(?phenotype, ?subjectID),
5: greaterThan(?phraseage, "33"^^long)
6: ->
7: AUTISMC1000033(?phenotype), ]
8: subject_has_quality_or_disposition(?subjectID, ?phenotype)
9:
10: AUTISMC000000(?a),
11: ADI_2003_acqorlossoflang_aphrase(?a, ?phraseage),
12: SubjectKey(?a, ?subjectID),
13: createOWLThing(?phenotype, ?subjectID),
14: greaterThan(?phraseage, "36"^^long)
15: ->
16: AUTISMC1000033(?phenotype),
17: subject_has_quality_or_disposition(?subjectID, ?phenotype)

```

Fonte: Elaborada pelo autor.

A única diferença entre as duas regras é o átomo com o predicado `greaterThan` em que a variável `?phraseage` é testada como maior que 33 na primeira regra, enquanto que, na segunda regra, essa variável é testada como maior que 36. Portanto, é possível concluir que a primeira regra subsume a segunda. Essa ontologia tem 152 regras, sendo que todas elas têm muitos átomos. O gerenciamento de um conjunto de regras como este pode ser muito complexo sem técnicas ou ferramentas que ajudem o usuário a identificar tais anomalias.

6.1.4.3 Ontologias mais relevantes

Nesta subseção são apresentados os resultados da identificação de anomalias em algumas ontologias mais relevantes. Essas ontologias foram consideradas relevantes pela quantidade de projetos que as utilizam no BioPortal. As 10 ontologias com mais projetos no BioPortal foram selecionadas em 30 de abril de 2015. Além dessas dez ontologias, as duas ontologia publicadas em artigos foram adicionadas ao grupo por se tratarem de projetos importantes e os únicos com regras. A lista a seguir mostra as ontologias mais relevantes selecionadas:

- *Gene Ontology*;
- *Ontology for Biomedical Investigations*;

- *Basic Formal Ontology*;
- *Systematized Nomenclature of Medicine - Clinical Terms*¹³;
- *Open Biological and Biomedical Ontologies Relationship Types*;
- *National Cancer Institute Thesaurus*;
- *Sequence Types and Features Ontology*;
- *Phenotypic Quality Ontology*;
- *Foundational Model of Anatomy*;
- *Chemical Entities of Biological Interest Ontology*;
- *Autism Phenologue* (YOUNG *et al.*, 2009);
- *Chemistry Ontology* (HASTINGS *et al.*, 2010).

Essas ontologias têm um total de 323.935 classes, 1.493 propriedades de dados, 512 propriedades de objetos e 253 regras. A Tabela 12 mostra as anomalias encontradas nessas ontologias. Dos 18 tipos de anomalias descritas neste trabalho, 9 foram encontrados nessas 12 ontologias. Este resultado mostra que mesmo ontologias importantes e muito utilizadas podem conter anomalias. A Tabela 12 mostra a ocorrência dos 9 tipos de anomalias: 3 tipos de deficiências (linhas 1, 5 e 6), 3 circularidades (linhas 2, 3 e 4), 2 redundâncias (linhas 8 e 9) e 1 inconsistência (linha 7).

Tabela 12 – Resultados das ontologias mais relevantes do Bioportal.

Anomalias	Total de ocorrências
<i>Chains of inheritance</i>	297.297
<i>Circularity between rules and taxonomy</i>	3
<i>Circular properties</i>	8
<i>Exact circularity in taxonomy</i>	2
<i>Lazy class or property</i>	240.983
<i>Lonely disjoint classes</i>	253
<i>Partition error in taxonomy</i>	460
<i>Redundancy by repetitive taxonomic definition</i>	1788
<i>Rule subsumption</i>	2

Fonte: Elaborada pelo autor.

¹³ Não incluída por não estar disponível para download, apenas para acesso online.

6.2 Estudo de caso com as visualizações de anomalias encontradas nas ontologias analisadas

A validação da visualização do sistema ONTO-Analyst é apresentada nesta seção através de um estudo de caso com visualizações de anomalias. Com exceção do tipo de anomalia *Redundant use of transitivity and symmetry* que é muito similar ao tipo *Redundant implication of transitivity or symmetry*, uma visualização de anomalia foi gerada para cada tipo de anomalia identificado na seção anterior. Para cada visualização, são mostradas as possíveis soluções dedutíveis a partir da mesma.

6.2.1 *Lonely disjoint classes*

A Figura 30 mostra uma ocorrência dessa anomalia na ontologia *Genomic Feature and Variation Ontology* (GFVO)¹⁴. Nessa visualização, a classe **Genotype** está isolada no ramo **FunctionalSpecification** das suas classes disjuntas, **Scaffold** e **Contig**, que pertencem ao ramo **Collection**. Sendo que, isso é um indicador de uma área mal projetada na ontologia.

Figura 30 – Visualização de uma ocorrência da anomalia *Lonely disjoint classes* na ontologia GFVO.

Fonte: Elaborada pelo autor.

Com a visualização, é possível encontrar classes isoladas e tomar a decisão de mudar ou não a ontologia. Como essa anomalia apenas destaca um design ruim, os desenvolvedores da ontologia podem decidir não alterá-la, mas isso pode se tornar um possível problema futuro. Neste caso em particular, uma possível solução é definir **Genotype** como uma subclasse de **Collection**, pois um Genótipo é uma coleção de informações.

6.2.2 *Partition error in taxonomy*

Na Figura 31 é mostrada uma ocorrência da anomalia *Partition error in taxonomy* na ontologia *Diagnostic Ontology* (DIAGONT)¹⁵. Nessa ocorrência da anomalia, a

¹⁴ Ontologia GFVO: <http://bioportal.bioontology.org/ontologies/GFVO>

¹⁵ <http://bioportal.bioontology.org/ontologies/DIAGONT>

classe `StoppingDetails` não pode ter nenhuma instância pois as instâncias não podem pertencer, ao mesmo tempo, a duas classes disjuntas.

Figura 31 – Visualização de uma ocorrência da anomalia *Partition error in taxonomy* na ontologia DIAGONT.

Fonte: Elaborada pelo autor.

A visualização pode ajudar os desenvolvedores da ontologia a identificar visualmente os relacionamentos que devem ser removidos/alterados. Eles podem focar na melhor solução para a anomalia utilizando as relações mostradas no gráfico, ao invés de ter que procurar conceitos problemáticos nas hierarquias de classes. Essa visualização se torna mais importante quando o erro é causado por uma ligação ou classe intermediária na hierarquia. No caso mostrado na Figura 31, qualquer subclasse ou relação de disjunção pode ser removida para a resolução da anomalia. A decisão de qual mudança deve ser feita, de forma a manter a intenção original dessa estrutura, fica a cargo dos desenvolvedores da ontologia DIAGONT.

6.2.3 *Exact circularity in taxonomy*

Na Figura 32 é mostrada uma ocorrência da anomalia *Exact circularity in taxonomy* na ontologia *Quantitative Imaging Biomarker Ontology* (QIBO)¹⁶. Nessa ontologia é encontrado um grupo de seis classes formando a circularidade apresentada. Na figura podem ser observados os seguintes alinhamentos:

- As classes `Magnetic_Resonance_Imaging` e `Microvascular_Permeability` são equivalentes;

¹⁶ <http://bioportal.bioontology.org/ontologies/QIBO>

- A classe `Magnetic_Resonance_Imaging` é superclasse de `Perfusion_MRI`;
- A classe `Microvascular_Permeability` é subclasse de `Functional_Measurement`;
- As classes `Perfusion_MRI` e `Functional_Measurement` são equivalentes.

Figura 32 – Visualização de uma ocorrência da anomalia *Exact circularity in taxonomy* na ontologia QIBO.

Fonte: Elaborada pelo autor.

Dessa forma, as 6 classes se tornam equivalentes. Entretanto, como existem as definições de herança, as subclasses deveriam conter algumas e não todas as instâncias de suas superclasses. Para corrigir essa situação, o desenvolvedor da ontologia pode analisar a visualização e decidir qual das relações de equivalência ou de herança é necessário remover. Caso as 6 classes realmente devam ser equivalentes, os desenvolvedores podem colocar todas essas classes em uma mesma superclasse e adicionar as relações de equivalências entre as 6 classes. Essas possíveis soluções são mais fáceis de serem identificadas com uma visualização.

6.2.4 Redundancy by repetitive taxonomic definition

Na ontologia *Semantic Web for Earth and Environment Technology Ontology* (SWEET)¹⁷ são encontradas várias ocorrências da anomalia *Redundancy by repetitive taxonomic definition*. Uma dessas ocorrências é mostrada na Figura 33. Essa ocorrência da anomalia é composta por três classes organizadas em herança. A classe `Solution` é superclasse de `Algorithm` e de `ParallelAlgorithm`, e a classe `Algorithm` é superclasse de `ParallelAlgorithm`.

¹⁷ Ontologia SWEET: <http://biportal.bioontology.org/ontologies/SWEET>

Figura 33 – Visualização de uma ocorrência da anomalia *Redundancy by repetitive taxonomic definition* na ontologia SWEET.

Fonte: Elaborada pelo autor.

Nessa ocorrência, a melhor solução para corrigir a anomalia é eliminar a definição de subclasse direta entre *Solution* e *ParallelAlgorithm*, pois *ParallelAlgorithm* é considerada uma subclasse de *Solution*, por meio da definição de herança entre *Solution* e *Algorithm*. Normalmente, as taxonomias são mostradas em forma de lista indentada (árvores) e quando ocorrem múltiplas superclasses, as classes filhas são repetidas dentro de todas as superclasses. A visualização em grafo pode facilitar a identificação da solução, pois pode mostrar todas as relações da taxonomia ao mesmo tempo.

6.2.5 *Lazy class or property*

Uma ocorrência da anomalia *Lazy class or property*, extraída da ontologia *Family Health History Ontology* (FHHO)¹⁸, é apresentada na Figura 34. Nessa figura, a anomalia ocorreu na propriedade de objeto *has_age_at_resolution* seguindo os critérios de ser uma folha na taxonomia, não ser usada por regras ou por restrições e não possuir instâncias.

Uma vantagem do método é identificar e visualizar características que seriam difíceis de procurar manualmente em uma ontologia. Com exceção da característica *isLeaf*, todas as demais características exigem um trabalho extra para serem descobertas manualmente.

¹⁸ <http://bioportal.bioontology.org/ontologies/FHHO>

Figura 34 – Visualização de uma ocorrência da anomalia *Lazy class or property* na ontologia FHHO.

Fonte: Elaborada pelo autor.

6.2.6 Chains of inheritance

Uma ocorrência da anomalia *Chains of inheritance*, extraída da ontologia FHHO (mesma ontologia da seção anterior), é apresentada na Figura 35. Nessa figura pode ser visto que a anomalia ocorreu nas definições de herança entre três classes que não são utilizadas em outras partes da ontologia. Com essa visualização, o desenvolvedor pode tomar a decisão de eliminar ou não classes que apresentam as características de não serem utilizadas por regras, restrições e instâncias.

6.2.7 Circular properties

A Figura 36 mostra uma ocorrência da anomalia *Circular properties* na ontologia ICF¹⁹. Nessa figura, são mostradas duas propriedades de objeto inversas: `primaryCareParent` e `primaryCareSubclasses`. Pode ser visto que ambas as propriedades têm a mesma classe de `rdfs:domain` (`LinearizationSection`) e a mesma classe de `rdfs:range` (`ICDSection`).

Um *reasoner* concluirá implicações, baseado nessas definições, que provavelmente não eram esperadas pelos desenvolvedores da ontologia. Por exemplo, assumindo que existam duas instâncias *x* e *y*, pertencendo respectivamente às classes `LinearizationSection` e `ICDSection`, e que a instância *x* tenha a propriedade `primarycareParent` com *y*, um *reasoner* irá concluir que *y* tem uma relação `primaryCareSubclasses` com *x*. Após essa conclusão, a instância *x* precisa ser classificada também como `ICDSection` e a instância *y*

¹⁹ Ontologia ICF: <http://bioportal.bioontology.org/ontologies/ICF>

Figura 35 – Visualização de uma ocorrência da anomalia *Chains of inheritance* na ontologia FHHO.

Fonte: Elaborada pelo autor.

Figura 36 – Visualização de uma ocorrência da anomalia *Circular Properties* na ontologia ICF.

Fonte: Elaborada pelo autor.

precisa ser classificada também como `LinearizationSection`, o que, provavelmente, não era o resultado esperado pelos desenvolvedores.

Além disso, essa ontologia tem o agravante de que a classe `LinearizationSection` é uma subclasse de `ICDSection`. Isso é um agravante, pois todas as instâncias que usam essas propriedades de objetos serão classificadas nas duas classes. Como todas as instâncias de uma classe pertencem também às suas superclasses, isso acarreta em uma circularidade. As implicações da circularidade provavelmente não são as pretendidas. Uma possível solução é mudar os `rdfs:range` das propriedades de objetos para `LinearizationSection`, pois todas as instâncias usadas por essas propriedades serão classificadas como tal (mudanças como essas podem melhorar a performance dos *reasoners*).

6.2.8 Circularity between rules and taxonomy

Uma ocorrência da anomalia *Circularity between rules and taxonomy*, extraída da ontologia *UnderwaterEnvironment*²⁰, é apresentada na Figura 37. Nessa figura, a circularidade ocorre na Regra Rule_15 em que o predicado `LowTemperature` do conseqüente é uma subclasse do predicado `Temperature` do antecedente. O mais interessante dessa visualização é que ela mostra apenas as partes envolvidas na anomalia.

Figura 37 – Visualização de uma ocorrência da anomalia *Circularity between rules and taxonomy* na ontologia *UnderwaterEnvironment*.

Fonte: Elaborada pelo autor.

Normalmente, as visualizações de regras SWRL mostram todos os seus átomos, tornando a visualização mais complexa. Nas visualizações propostas no sistema ONTO-Analyst são omitidas as partes que não fazem parte da anomalia tornando a visualização da anomalia mais clara e objetiva.

Nesse caso, a melhor solução é que a definição de uma `LowTemperature` seja definida diretamente nas restrições dessa classe. O problema em usar uma regra para fazer essa classificação é que instâncias podem ser declaradas como pertencentes a essa classe, mesmo que não cumpram as condições da regra. Já se a classe tiver a restrição em sua definição, não podem existir instâncias dela que não atendam a restrição.

6.2.9 Redundant derivation in the antecedent

A Figura 38 mostra a visualização de uma ocorrência da anomalia *Redundant derivation in the antecedent* na ontologia *ContextOntology*²¹. Essa ocorrência é composta

²⁰ Ontologia *UnderwaterEnvironment*: <https://goo.gl/D4J1SK>

²¹ Ontologia *ContextOntology*: <https://goo.gl/5NidDu>

por cinco classes, uma regra e dois átomos de antecedentes (existem mais átomos nessa regra, mas apenas os relacionados com a anomalia são visualizados).

Figura 38 – Visualização de uma ocorrência da anomalia *Redundant Derivation in the Antecedent* na ontologia *ContextOntology*.

Fonte: Elaborada pelo autor.

O átomo `context-element(?lightRef)` referencia todas as instâncias que pertencem a classe `context-element` e o átomo `light-sensor(?lightRef)` referencia todas as instâncias da classe `light-sensor`. As instâncias de `light-sensor` são também instâncias de `context-element`, tendo em vista que `context-element` é uma superclasse de `light-sensor`. Portanto, o átomo `context-element(?lightRef)` não é necessário para essa regra, o que caracteriza a redundância.

Para essa ocorrência em particular, existem duas possíveis soluções. A primeira é revisar os relacionamentos hierárquicos dessas duas classes. A segunda é remover o átomo redundante da regra. Para aplicar uma dessas soluções é necessária uma análise dos desenvolvedores de forma a manter a intenção original da ontologia. A visualização gráfica dessa anomalia torna mais fácil o entendimento e a decisão de alteração das estruturas da ontologia.

6.2.10 Redundant implication of transitivity or symmetry

A Figura 39 mostra a visualização de uma ocorrência da anomalia *Redundant implication of transitivity or symmetry* na ontologia *Rules for processing graph knowledge*.

*base*²². Essa ocorrência é composta por uma regra, dois átomos do antecedente, um átomo do conseqüente e uma propriedade transitiva. A propriedade transitiva é usada pelos três átomos, sendo que o átomo do conseqüente é uma redundância e pode ser retirado da regra em questão. A principal vantagem da visualização é que a propriedade `isBefore` é destacada como transitiva na visualização e isso torna mais fácil a tomada de decisão para alterar a ontologia. As possíveis soluções para essa anomalia são remover o átomo do conseqüente ou remover a informação de que `isBefore` é uma propriedade transitiva.

Figura 39 – Visualização de uma ocorrência da anomalia *Redundant implication of transitivity or symmetry* na ontologia *Rules for processing graph knowledgebase*.

Fonte: Elaborada pelo autor.

6.2.11 Rule subsumption

A Figura 40 mostra a visualização de uma ocorrência da anomalia *Rule subsumption* na ontologia elaborada por Young *et al.* (2009). Nessa visualização, a regra `Rule_92` subsume a regra `Rule_119`. Nesse caso, como a consulta SPARQL que identifica essa anomalia é simples, a visualização acaba sendo limitada, pois omite os átomos que estão envolvidos na relação de subsunção. Esse é um exemplo de limitação de uma visualização gerada de forma automática, a partir de uma consulta SPARQL e um resultado selecionado.

6.3 Considerações finais

Ao longo deste capítulo, foram apresentadas análises quantitativas da identificação de anomalias e qualitativas com o estudo de caso sobre visualizações dos tipos de anomalias encontrados pelo método ONTO-Analyst.

²² Ontologia *Rules for processing graph knowledgebase*: <http://ontology.dumontierlab.com/graph-rules.owl>

Figura 40 – Visualização de uma ocorrência da anomalia *Rule subsumption* na ontologia elaborada por Young *et al.* (2009).

Fonte: Elaborada pelo autor.

Para as análises quantitativas, foi realizada a seleção dos repositórios públicos de ontologias, geração de estatísticas sobre as ontologias escolhidas, análise da conversão dessas para o formato MetaFOR e análise da identificação das anomalias. 608 ontologias, de 4 repositórios e 2 artigos científicos, foram convertidas para o formato MetaFOR. Essas ontologias contêm 4.377.570 classes, 23.593 propriedades de objetos, 8.487 propriedades de dados, 979 regras e 7 restrições de cardinalidades, que foram analisadas pelo ONTO-Analyst.

O ONTO-Analyst, usando as ontologias no formato MetaFOR, testou 18 tipos de anomalias e encontrou ocorrências de 12 tipos. A análise quantitativa comprovou que o método ONTO-Analyst é capaz de identificar automaticamente anomalias que ocorrem em diferentes partes de uma ontologia.

Ainda nas análises quantitativas, foi realizada uma verificação das 10 ontologias mais importantes do BioPortal e nas 2 ontologias dos artigos, mostrando que, mesmo em ontologias muito importantes, anomalias são encontradas. Foram encontrados 9 tipos de anomalias nessas ontologias importantes.

Na análise qualitativa com o estudo de caso, foram escolhidos 11 exemplos de anomalia de tipos diferentes do total encontrado nas 608 ontologias analisadas. Para cada um, foi apresentada a visualização e mostradas as possíveis soluções deduzíveis a partir das visualizações.

CONCLUSÕES

O objetivo principal deste trabalho foi propor um método para identificar ocorrências de anomalias em ontologias, chamado ONTO-Analyst, que permite a desenvolvedores identificar essas anomalias usando consultas SPARQL e visualizá-las de forma gráfica. Esse método propõe o uso de uma ontologia, a MetaFOR, para descrever a estrutura de uma outra ontologia e usa consultas SPARQL para identificar anomalias nessa descrição. Uma vez identificadas, as anomalias podem ser apresentadas, de forma visual, em grafos com nós representando as entidades envolvidas na anomalia e arestas representando as relações dessas entidades.

Além disso, um protótipo, o sistema ONTO-Analyst, foi desenvolvido usando esse método para demonstrar a sua eficácia. O protótipo lê uma ontologia em OWL, cria uma descrição da sua estrutura, usando a ontologia MetaFOR, aplica um conjunto de consultas SPARQL para encontrar anomalias, e gera uma visualização para as anomalias encontradas. Como o ONTO-Analyst usa a linguagem SPARQL para encontrar anomalias, ele permite que os próprios desenvolvedores escrevam consultas para encontrar novos tipos de anomalias ou detectar estruturas que não seguem as melhores práticas, escolhidas para um projeto.

A identificação de anomalias, usando a ontologia MetaFOR para descrever informações estruturais, traz três importantes vantagens aos desenvolvedores de ontologias:

- A facilidade de utilizar recursos que eles conhecem, tais como: ontologias, consultas SPARQL e regras SWRL;
- A possibilidade de alterar, caso necessário, os critérios usados na identificação de anomalias. Como essa identificação é feita por consultas SPARQL e/ou regras SWRL, eles podem fazer alterações nessas consultas, pois elas são escritas em linguagens bem conhecidas na comunidade da Web Semântica;

- A facilidade de adicionar a identificação de outros tipos de anomalias, ou qualquer outro tipo de padrão estrutural que eles considerem importantes, na descrição estrutural da ontologia em MetaFOR. Para isso, eles devem apenas escrever novas consultas em SPARQL ou SWRL.

Outro ponto importante, desenvolvido no método, foi a visualização que funciona em colaboração com a identificação. Para cada identificação de anomalia escrita em uma consulta SPARQL, o sistema ONTO-Analyst gera uma visualização com o intuito de facilitar a resolução da anomalia. Para tanto, a visualização mostra somente as partes envolvidas na anomalia. Essa é uma importante característica, pois, normalmente, as visualizações de ontologias são gerais e apresentam todas as partes da ontologia. Ao visualizar somente os elementos que fazem parte da anomalia, o desenvolvedor pode focar na solução da mesma.

Para validar o método ONTO-Analyst e a ontologia MetaFOR foi conduzida uma série de experimentos com o protótipo desenvolvido, relacionados à identificação de anomalias e à visualização das suas ocorrências.

Foi criado um grupo representativo das ontologias públicas OWL usadas atualmente, que possui 608 ontologias OWL de 3 repositórios públicos de ontologias, de 2 artigos e do repositório GitHub. Todas as 608 ontologias foram convertidas para o formato MetaFOR. Elas continham 4.377.570 classes, 23.593 propriedades de objetos, 8.487 propriedades de dados, 979 regras e 7 restrições de cardinalidades. Usando esses dados, o sistema ONTO-Analyst procurou por 18 tipos de anomalias diferentes, todos retirados da literatura científica, encontrando 4.422.188 ocorrências de 12 tipos de anomalias. De um total de 3 tipos de circularidades, 7 redundâncias, 5 inconsistências e 3 deficiências verificadas pelo ONTO-Analyst, foram encontrados 3 tipos de circularidades, 5 redundâncias, 1 inconsistência e 3 deficiências nas ontologias analisadas.

A análise quantitativa comprovou que o método ONTO-Analyst é capaz de identificar automaticamente os quatro diferentes tipos de anomalias (inconsistências, circularidades, redundâncias e deficiências) que ocorrem em diferentes partes de uma ontologia (classes, propriedades, regras e cardinalidades).

Além disso, foi possível determinar que:

- Inconsistências são bem verificadas pelos *reasoners*, a principal ferramenta de teste de ontologias usada por desenvolvedores, já que o número de inconsistências encontradas foi muito baixo. As deficiências, circularidades e redundâncias são mais comuns porque as ferramentas que são utilizadas pelos desenvolvedores de ontologia não se concentram na identificação desses tipos de problemas;
- Existem muitas classes nas ontologias analisadas (em relação à propriedades e ins-

tâncias), por isso, todos os tipos de anomalias que podem ocorrer em classes (ou propriedades) foram encontradas;

- Regras SWRL não são muito comuns em ontologias e conjuntos de regras geralmente são pequenos e bem construídos. Apenas 5 tipos de anomalias, dos 9 tipos verificados que envolvem regras SWRL, foram encontrados;
- O ONTO-Analyst permite a identificação de anomalias em regras que não são disparadas por *reasoners*, por não terem seus requisitos de disparo atendidos, sendo essa uma importante contribuição;
- Restrições de cardinalidade são muito raras em ontologias. Apenas 7 definições de cardinalidades foram encontradas nas ontologias analisadas;
- Mesmo restringindo o conjunto de ontologias às 10 mais usadas em projetos do BioPortal e às 2 ontologias encontradas a partir de artigos, foram encontrados 9 tipos de anomalias dos 18 analisadas, o que demonstra que mesmo ontologias importantes e muito usadas podem se beneficiar deste método.

Dada a importância dos 4 repositórios analisados e do amplo escopo coberto pelas 608 ontologias, os resultados dos experimentos confirmam a eficiência do ONTO-Analyst e a necessidade de ferramentas como ele.

Após as análises quantitativas, foi realizado um estudo de caso das visualizações geradas para ocorrências de 11 tipos de anomalias encontradas. Foi apresentada uma ocorrência de cada tipo de anomalia encontrada, com exceção à anomalia *Redundant use of transitivity and symmetry*. Para cada visualização, foram discutidas as possíveis soluções para as anomalias mostradas. Foi possível mostrar que uma das grandes vantagens do uso das visualizações é que elas destacam as entidades relacionadas a uma anomalia, facilitando a sua resolução.

Para facilitar o rápido entendimento das visualizações, foram desenvolvidos vários símbolos para representar os diferentes tipos de entidades e os diferentes tipos de relações entre as mesmas. Outra característica importante, adotada nas visualizações, foi a representação de hierarquias na vertical. Apesar de usar um grafo para representar elementos de uma ontologia, o ONTO-Analyst sempre mantém a representação da taxonomia na vertical, claramente determinando visualmente as relações de herança entre as entidades.

7.1 Contribuições

Esta tese de doutorado gerou as seguintes contribuições científicas e tecnológicas:

- Elaboração da MetaFOR, uma ontologia sobre ontologias, que é usada para descrever a estrutura de uma ontologia. Ela é usada, em conjunto com consultas SPARQL, no ONTO-Analyst para identificar ocorrências de anomalias;
- Implementação de um conjunto de consultas SPARQL que, quando executadas em ontologias no formato MetaFOR, identificam as ocorrências de anomalias nessas ontologias. Essas consultas permitem que as identificações sejam extensíveis/modificáveis. Desenvolvedores de ontologias podem alterar as identificações existentes ou inserir novas identificações com a criação de novas regras SWRL ou novas consultas SPARQL;
- Elaboração do protótipo do sistema ONTO-Analyst para extração automática de informações de uma ontologia, geração de uma descrição no formato MetaFOR e detecção de anomalias, usando consultas SPARQL. Além disso, o protótipo permite a adição de novas estruturas, não previstas inicialmente no formato da MetaFOR, por meio da implementação de interfaces Java;
- Adição, ao protótipo, da visualização das ocorrências de anomalias encontradas para facilitar o entendimento das mesmas;
- Análise quantitativa da real ocorrência de anomalias em 608 ontologias públicas e a apresentação de quais são as ocorrências mais comuns em ontologias. No total, foram escolhidos 18 tipos diferentes de anomalias, dos quais 12 foram encontrados nas ontologias analisadas;
- Estudo de caso das visualizações geradas de ocorrências de anomalias encontradas em ontologias públicas, mostrando que a visualização permite identificar mais facilmente possíveis soluções para anomalias;
- Dois artigos publicados em conferências, um capítulo de livro e dois artigos submetidos para periódicos.

7.2 Trabalhos futuros

Como trabalhos futuros para podem ser citados:

- Elaboração de um ambiente Web, similar ao ONTO-Analyst, para identificação e visualização de anomalias;
- Elaboração de um ambiente interativo para dar suporte e facilitar o desenvolvimento de consultas SPARQL pelos desenvolvedores de ontologias que forem usar o ONTO-Analyst. Atualmente essas consultas devem ser criadas em uma simples caixa de texto no ONTO-Analyst;

- Elaboração de experimentos com desenvolvedores de ontologias em que eles desenvolvam suas próprias consultas para identificar seus padrões de interesse;
- Elaboração e aplicação de questionários de avaliação qualitativa do método com desenvolvedores de ontologias. Dois diferentes questionários poderiam ser aplicados:
 - Questionário relacionado à identificação de anomalias, buscando a opinião sobre a relevância das estruturas analisadas;
 - Questionário relacionado aos ganhos e às deficiências que a visualização de ocorrências de anomalias traz.
- Implementação de outros grupos de anomalias ou grupos de padrões que podem ser interessantes aos desenvolvedores de ontologias, como *Ontology Design Patterns* (ODP);
- Integrar o sistema ONTO-Analyst como um plug-in para o editor Protégé. Isso poderá aumentar muito a base de usuários para a ferramenta.

REFERÊNCIAS

ALANI, H. TGVizTab: An Ontology Visualisation Extension for Protege. In: **Proceedings of Knowledge Capture (K-Cap'03), Workshop on Visualization Information in Knowledge Engineering**. Sanibel Island, Florida, USA: [s.n.], 2003. p. 23–26. Citado na página 78.

ANTONIOU, G.; HARMELEN, F. van. **A Semantic Web Primer, 2nd Edition (Cooperative Information Systems)**. 2. ed. [S.l.]: The MIT Press, 2008. ISBN 0-262-01242-1. Citado nas páginas 35, 38 e 41.

BAEHRECKE, E. H.; DANG, N.; BABARIA, K.; SHNEIDERMAN, B. Visualization and analysis of microarray and gene ontology data with treemaps. **BMC Bioinformatics**, v. 5, n. 1, p. 84, 2004. ISSN 1471-2105. Disponível em: <<http://dx.doi.org/10.1186/1471-2105-5-84>>. Citado nas páginas 76 e 78.

BAUMEISTER, J.; SEIPEL, D. Anomalies in ontologies with rules. **Web Semantics: Science, Services and Agents on the World Wide Web**, v. 8, n. 1, p. 55–68, mar. 2010. ISSN 1570-8268. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S1570826809000778>>. Citado nas páginas 49, 50, 51, 52, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68 e 69.

BECHHOFFER, S.; HARMELEN, F. van; HENDLER, J.; HORROCKS, I.; MCGUINNESS, D. L.; PATEL-SCHNEIDER, P. F.; STEIN, L. A. **OWL Web Ontology Language Reference**. <http://www.w3.org/TR/owl-ref/>, 2004. Citado nas páginas 29, 39 e 41.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The Semantic Web. **Scientific American**, p. 29–37, maio 2001. Citado nas páginas 29, 36 e 37.

BOSTOCK, M.; OGIEVETSKY, V.; HEER, J. D3: Data-Driven Documents. **IEEE Transactions on Visualization and Computer Graphics**, v. 17, n. 12, p. 2301–2309, dez. 2011. ISSN 1077-2626. Citado na página 104.

BRULS, M.; HUIZING, K.; WIJK, J. J. van. Squarified Treemaps. In: LEEUW, W. C. de; LIERE, R. van (Ed.). **Data Visualization 2000: Proceedings of the Joint EUROGRAPHICS and IEEE TCVG Symposium on Visualization in Amsterdam, The Netherlands, May 29–30, 2000**. Vienna: Springer Vienna, 2000. p. 33–42. ISBN 978-3-7091-6783-0. DOI: 10.1007/978-3-7091-6783-0_4. Disponível em: <http://dx.doi.org/10.1007/978-3-7091-6783-0_4>. Citado na página 73.

CORCHO, O.; ROUSSEY, C.; BLÁZQUEZ, L. M. V.; PÉREZ, I. Pattern-based OWL Ontology Debugging Guidelines. In: **Proceedings of the 2009 International Conference on Ontology Patterns - Volume 516**. Aachen, Germany, Germany: CEUR-WS.org, 2009. (WOP'09), p. 68–82. Disponível em: <<http://dl.acm.org/citation.cfm?id=2889761.2889767>>. Citado na página 57.

DAGA, E.; BLOMQVIST, E.; GANGEMI, A.; MONTIEL, E.; NIKITINA, N.; PRESUTTI, V.; VILLAZÓNTERRAZAS, B. **NeOn D2.5.2 Pattern based ontology design: methodology and software support**. [S.l.], 2010. Citado nas páginas 56, 57 e 58.

D'AQUIN, M.; NOY, N. F. Where to publish and find ontologies? A survey of ontology libraries. **Web Semantics: Science, Services and Agents on the World Wide Web**, v. 11, p. 96–111, mar. 2012. ISSN 1570-8268. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S157082681100076X>>. Citado na página 108.

DONG, J. S.; FENG, Y.; LI, Y. F.; SUN, J. A tools environment for developing and reasoning about ontologies. In: **12th Asia-Pacific Software Engineering Conference (APSEC'05)**. [S.l.: s.n.], 2005. p. 465–472. Citado na página 54.

DUDÁŠ, M.; ZAMAZAL, O.; SVÁTEK, V. Roadmapping and Navigating in the Ontology Visualization Landscape. In: JANOWICZ, K.; SCHLOBACH, S.; LAMBRIX, P.; HYVÖNEN, E. (Ed.). **Knowledge Engineering and Knowledge Management: 19th International Conference, EKAW 2014, Linköping, Sweden, November 24-28, 2014. Proceedings**. Cham: Springer International Publishing, 2014. p. 137–152. ISBN 978-3-319-13704-9. Disponível em: <http://dx.doi.org/10.1007/978-3-319-13704-9_11>. Citado nas páginas 71, 72, 73 e 76.

FAHAD, M.; QADIR, M. A. A Framework for Ontology Evaluation. In: **Proceedings of the 16th International Conference on Conceptual Structures**. Toulouse, France: [s.n.], 2008. v. 354, p. 149–158. Citado nas páginas 49, 51, 57, 59, 60 e 61.

FAHAD, M.; QADIR, M. A.; SHAH, S. A. H. Evaluation of Ontologies and DL Reasoners. In: SHI, Z.; MERCIER-LAURENT, E.; LEAKE, D. (Ed.). **Intelligent Information Processing IV**. Springer US, 2008, (IFIP – The International Federation for Information Processing, 288). p. 17–27. ISBN 978-0-387-87684-9 978-0-387-87685-6. DOI: 10.1007/978-0-387-87685-6_5. Disponível em: <http://link.springer.com/chapter/10.1007/978-0-387-87685-6_5>. Citado na página 51.

FEIGENBAUM, L.; HERMAN, I.; HONGSERMEIER, T.; NEUMANN, E.; STEPHENS, S. The Semantic Web In Action. **Scientific American**, n. December 2007, p. 64–71, 2007. Disponível em: <<http://www.scientificamerican.com/article.cfm?id=the-semantic-web-in-action>>. Citado na página 38.

FENG, Y.; LIU, Y.; LI, Y.-F.; ZHANG, D. Discovering Anomalies in Semantic Web Rules. In: **2010 Fourth International Conference on Secure Software Integration and Reliability Improvement (SSIRI)**. [S.l.: s.n.], 2010. p. 33–42. Citado nas páginas 54, 58, 65, 66, 67, 68 e 69.

FU, B.; NOY, N. F.; STOREY, M.-A. Indented Tree or Graph? A Usability Study of Ontology Visualization Techniques in the Context of Class Mapping Evaluation. In: ALANI, H.; KAGAL, L.; FOKOUE, A.; GROTH, P.; BIEMANN, C.; PARREIRA, J. X.; AROYO, L.; NOY, N.; WELTY, C.; JANOWICZ, K. (Ed.). **The Semantic Web – ISWC 2013: 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 117–134. ISBN 978-3-642-41335-3. Disponível em: <http://dx.doi.org/10.1007/978-3-642-41335-3_8>. Citado nas páginas 71 e 72.

GEROIMENKO, V.; CHEN, C. **Visualizing the Semantic Web: XML-based Internet and Information Visualization**. Springer, 2006. ISBN 978-1-85233-976-0. Disponível em: <<https://books.google.com.br/books?id=GPFpBdcDXdYC>>. Citado na página 71.

GÓMEZ-PÉREZ, A. Evaluation of Taxonomic Knowledge in Ontologies and Knowledge Bases. In: **Proceedings of the 12th Banff Knowledge Acquisition for Knowledge-Based Systems Worksho**. Banff, Alberta, Canada: [s.n.], 1999. p. 16–21. Citado nas páginas 50, 54, 57, 58, 61 e 62.

_____. Evaluation of ontologies. **International Journal of Intelligent Systems**, v. 16, n. 3, p. 391–409, mar. 2001. ISSN 1098-111X. Disponível em: <[http://onlinelibrary.wiley.com/doi/10.1002/1098-111X\(200103\)16:3<391::AID-INT1014>3.0.CO;2-2/abstract](http://onlinelibrary.wiley.com/doi/10.1002/1098-111X(200103)16:3<391::AID-INT1014>3.0.CO;2-2/abstract)>. Citado na página 57.

GÓMEZ-PÉREZ, A.; JURISTO, N.; PAZOS, J. Evaluation and Assessment of the Knowledge Sharing Technology. In: **Towards very large knowledge bases**. Holanda: IOS Press, 1995. p. 289–296. ISBN 90-5199-217-3. Citado nas páginas 43 e 44.

GRIMM, S.; WISSMANN, J. Elimination of Redundancy in Ontologies. In: **The Semantic Web: Research and Applications**. Springer, Berlin, Heidelberg, 2011. p. 260–274. Disponível em: <https://link.springer.com/chapter/10.1007/978-3-642-21034-1_18>. Citado nas páginas 50, 55, 56 e 58.

GRUBER, T. R. A Translation Approach to Portable Ontology Specifications. **Knowl. Acquis.**, v. 5, n. 2, p. 199–220, jun. 1993. ISSN 1042-8143. Disponível em: <<http://dx.doi.org/10.1006/knac.1993.1008>>. Citado na página 36.

GUO, S. S.; CHAN, C. W. A Comparison and Analysis of Some Ontology Visualization Tools. In: **SEKE**. Knowledge Systems Institute Graduate School, 2011. p. 357–362. ISBN 1-891706-29-2. Disponível em: <<http://dblp.uni-trier.de/db/conf/seke/seke2011.html#GuoC11>>. Citado nas páginas 72 e 76.

HAM, F. V.; WIJK, J. J. V. Beamtrees: compact visualization of large hierarchies. In: **IEEE Symposium on Information Visualization, 2002. INFOVIS 2002**. [S.l.: s.n.], 2002. p. 93–100. Citado nas páginas 73 e 77.

HASSANPOUR, S.; O'CONNOR, M. J.; DAS, A. K. Exploration of SWRL Rule Bases through Visualization, Paraphrasing, and Categorization of Rules. In: GOVERNATORI, G.; HALL, J.; PASCHKE, A. (Ed.). **Rule Interchange and Applications: International Symposium, RuleML 2009, Las Vegas, Nevada, USA, November 5-7, 2009. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 246–261. ISBN 978-3-642-04985-9. Disponível em: <http://dx.doi.org/10.1007/978-3-642-04985-9_23>. Citado na página 45.

HASTINGS, J.; DUMONTIER, M.; HULL, D.; HORRIDGE, M.; STEINBECK, C.; SATTLER, U.; STEVENS, R.; HÖRNE, T.; BRITZ, K. Representing Chemicals using OWL, Description Graphs and Rules. In: **Proceedings of the 7th International Workshop on OWL: Experiences and Directions (OWLED 2010)**. [S.l.: s.n.], 2010. p. 1–10. Citado nas páginas 109 e 125.

HEFLIN, J. **OWL Web Ontology Language - Use Cases and Requirements**. [S.l.], 2004. [Http://www.w3.org/TR/2004/REC-webont-req-20040210/](http://www.w3.org/TR/2004/REC-webont-req-20040210/). Disponível em: [<http://www.w3.org/TR/2004/REC-webont-req-20040210/>](http://www.w3.org/TR/2004/REC-webont-req-20040210/). Citado na página 36.

HOGAN, A.; HARTH, A.; PASSANT, A.; DECKER, S.; POLLERES, A. Weaving the Pedantic Web. In: BIZER, C.; HEATH, T.; BERNERS-LEE, T.; HAUSENBLAS, M. (Ed.). **LDOW**. CEUR-WS.org, 2010. (CEUR Workshop Proceedings, v. 628). Disponível em: <http://dblp.uni-trier.de/db/conf/www/ldow2010.html#HoganHPDP10>. Citado na página 57.

HORRIDGE, M.; BECHHOFFER, S. The OWL API: A Java API for OWL Ontologies. **Semant. web**, v. 2, n. 1, p. 11–21, jan. 2011. ISSN 1570-0844. Disponível em: <http://dl.acm.org/citation.cfm?id=2019470.2019471>. Citado na página 90.

HORROCKS, I.; PATEL-SCHNEIDER, P. F.; BOLEY, H.; TABET, S.; GROSOFF, B.; DEAN, M. **SWRL: A Semantic Web Rule Language Combining OWL and RuleML**. [S.l.], 2004. Disponível em: <http://www.w3.org/Submission/SWRL>. Citado na página 44.

KATIFORI, A.; HALATSIS, C.; LEPOURAS, G.; VASSILAKIS, C.; GIANNOPOULOU, E. Ontology Visualization Methods-A Survey. **ACM Comput. Surv.**, v. 39, n. 4, p. 10, nov. 2007. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/1287620.1287621>. Citado nas páginas 72, 73, 74, 75, 76, 77 e 79.

KATIFORI, A.; TOROU, E.; HALATSIS, C.; LEPOURAS, G.; VASSILAKIS, C. A Comparative Study of Four Ontology Visualization Techniques in Protege: Experiment Setup and Preliminary Results. In: **Proceedings of the Conference on Information Visualization**. Washington, DC, USA: IEEE Computer Society, 2006. (IV '06), p. 417–423. ISBN 0-7695-2602-0. Disponível em: <http://dx.doi.org/10.1109/IV.2006.3>. Citado nas páginas 72, 75 e 78.

KEET, C. M.; SUÁREZ-FIGUEROA, M. C.; POVEDA-VILLALÓN, M. The Current Landscape of Pitfalls in Ontologies. In: **KEOD 2013 - Proceedings of the International Conference on Knowledge Engineering and Ontology Development, Vilamoura, Algarve, Portugal, 19-22 September, 2013**. [s.n.], 2013. p. 132–139. Disponível em: <http://dx.doi.org/10.5220/0004517901320139>. Citado na página 53.

KHONDOKER, M. R.; MUELLER, P. Comparing Ontology Development Tools Based on an Online Survey. In: AO, S. I.; GELMAN, L.; HUKINS, D. W.; HUNTER, A.; KORSUNSKY, A. M. (Ed.). **Proceedings of the World Congress on Engineering 2010 Vol I, WCE '10, June 30 - July 2, 2010, London, U.K.** [S.l.]: Newswood Limited, 2010. (Lecture Notes in Engineering and Computer Science), p. 188–192. ISBN 978-988-17012-9-9. Citado na página 104.

KIFER, M. Rule Interchange Format: The Framework. In: CALVANESE, D.; LAUSEN, G. (Ed.). **Web Reasoning and Rule Systems: Second International Conference, RR 2008, Karlsruhe, Germany, October 31-November 1, 2008. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 1–11. ISBN 978-3-540-88737-9. Disponível em: http://dx.doi.org/10.1007/978-3-540-88737-9_1. Citado na página 46.

- KLYNE, G.; CARROLL, J. J. **Resource Description Framework (RDF): Concepts and Abstract Syntax**. [S.l.], 2004. Disponível em: <<https://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>>. Citado nas páginas 37 e 38.
- LANZENBERGER, M.; SAMPSON, J.; RESTER, M. Visualization in Ontology Tools. In: **Complex, Intelligent and Software Intensive Systems, 2009. CISIS '09. International Conference on**. [S.l.: s.n.], 2009. p. 705–711. Citado nas páginas 71 e 72.
- LI, K.; GAO, W.; ZHANG, H.; LIU, X.; SHAO, S. EDS2dcc: A Method of Eliminating Direct Super-class to Detect Cycle of Concepts. In: **2009 Second International Workshop on Knowledge Discovery and Data Mining**. [S.l.: s.n.], 2009. p. 598–601. Citado nas páginas 54 e 58.
- LOHMANN, S.; NEGRU, S.; HAAG, F.; ERTL, T. VOWL 2: User-Oriented Visualization of Ontologies. In: JANOWICZ, K.; SCHLOBACH, S.; LAMBRIX, P.; HYVÖNEN, E. (Ed.). **Knowledge Engineering and Knowledge Management: 19th International Conference, EKAW 2014, Linköping, Sweden, November 24–28, 2014. Proceedings**. Cham: Springer International Publishing, 2014. p. 266–281. ISBN 978-3-319-13704-9. DOI: 10.1007/978-3-319-13704-9_21. Disponível em: <http://dx.doi.org/10.1007/978-3-319-13704-9_21>. Citado nas páginas 71, 74 e 75.
- MAZZA, R. **Introduction to Information Visualization**. 1. ed. [S.l.]: Springer Publishing Company, Incorporated, 2009. ISBN 1-84800-218-1 978-1-84800-218-0. Citado na página 72.
- MCGUINNESS, D. L.; HARMELEN, F. V. **OWL Web Ontology Language Overview**. [S.l.], 2004. Citado na página 39.
- MOTIK, B.; GRAU, B. C.; HORROCKS, I.; WU, Z.; FOKOUE, A.; LUTZ, C. **OWL 2 Web Ontology Language Profiles (Second Edition)**. [S.l.], 2012. Disponível em: <<https://www.w3.org/TR/owl2-profiles/>>. Citado nas páginas 39 e 40.
- MOTIK, B.; PATEL-SCHNEIDER, P. F.; PARSIA, B.; BOCK, C.; FOKOUE, A.; HAASE, P.; HOEKSTRA, R.; HORROCKS, I.; RUTTENBERG, A.; SATTTLER, U.; SMITH, M. **OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax**. [S.l.], 2009. Disponível em: <<https://www.w3.org/TR/2009/REC-owl2-syntax-20091027/>>. Citado na página 55.
- NOY, N.; MCGUINNESS, D. **Ontology development 101: A guide to creating your first ontology**. [S.l.], 2001. Disponível em: <<http://www.ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness-abstract.html>>. Citado nas páginas 36, 37 e 57.
- NOY, N. F.; FERGERSON, R. W.; MUSEN, M. A. The Knowledge Model of Protégé-2000: Combining Interoperability and Flexibility. In: DIENG, R.; CORBY, O. (Ed.). **Knowledge Engineering and Knowledge Management Methods, Models, and Tools: 12th International Conference, EKAW 2000 Juan-les-Pins, France, October 2–6, 2000 Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000. p. 17–32. ISBN 978-3-540-39967-4. DOI: 10.1007/3-540-39967-4_2. Disponível em: <http://dx.doi.org/10.1007/3-540-39967-4_2>. Citado na página 75.

ORLANDO, J. P.; MUSEN, M. A.; MOREIRA, D. A. User Extensible System to Identify Problems in OWL Ontologies and SWRL Rules. In: BASSILIADES, N.; GOTTLÖB, G.; SADRI, F.; PASCHKE, A.; ROMAN, D. (Ed.). **Rule Technologies: Foundations, Tools, and Applications: 9th International Symposium, RuleML 2015, Berlin, Germany, August 2-5, 2015, Proceedings**. Cham: Springer International Publishing, 2015. p. 112–126. ISBN 978-3-319-21542-6. Disponível em: <http://dx.doi.org/10.1007/978-3-319-21542-6_8>. Citado nas páginas 44, 50 e 103.

ORLANDO, J. P.; RÍVOLLI, A.; MOREIRA, D. A. Introdução ao desenvolvimento colaborativo de regras SWRL com o SWRL Editor. In: **TÓPICOS EM MULTIMÍDIA, HIPERMÍDIA E WEB**. [S.l.]: Sociedade Brasileira de Computação, 2012. p. 1–30. ISBN 978-85-7669-265-2. Citado na página 44.

PAK, J.; ZHOU, L. A Framework for Ontology Evaluation. In: SHARMAN, R.; RAO, H. R.; RAGHU, T. S. (Ed.). **Exploring the Grand Challenges for Next Generation E-Business**. Springer Berlin Heidelberg, 2009, (Lecture Notes in Business Information Processing, 52). p. 10–18. ISBN 978-3-642-17448-3 978-3-642-17449-0. DOI: 10.1007/978-3-642-17449-0_2. Disponível em: <http://link.springer.com/chapter/10.1007/978-3-642-17449-0_2>. Citado nas páginas 30, 49 e 57.

PLAISANT, C.; GROSJEAN, J.; BEDERSON, B. B. SpaceTree: Supporting Exploration in Large Node Link Tree, Design Evolution and Empirical Evaluation. In: **Proceedings of the IEEE Symposium on Information Visualization (InfoVis'02)**. Washington, DC, USA: IEEE Computer Society, 2002. (INFOVIS '02), p. 57–64. ISBN 0-7695-1751-X. Disponível em: <<http://dl.acm.org/citation.cfm?id=857191.857752>>. Citado nas páginas 73 e 77.

POVEDA-VILLALÓN, M. **Ontology Evaluation: a pitfall-based approach to ontology diagnosis**. Tese (Ph.D. Thesis) — Universidad Politécnica de Madrid, Madrid, 2016. Disponível em: <<http://oa.upm.es/39448/>>. Citado nas páginas 43, 53 e 58.

POVEDA-VILLALÓN, M.; GÓMEZ-PÉREZ, A.; SUÁREZ-FIGUEROA, M. C. OOPS! (Ontology Pitfall Scanner!): An On-line Tool for Ontology Evaluation. **Int. J. Semant. Web Inf. Syst.**, v. 10, n. 2, p. 7–34, abr. 2014. ISSN 1552-6283. Disponível em: <<http://dx.doi.org.ez67.periodicos.capes.gov.br/10.4018/ijswis.2014040102>>. Citado na página 53.

POVEDA-VILLALÓN, M.; SUÁREZ-FIGUEROA, M. C.; GÓMEZ-PÉREZ, A. Validating Ontologies with OOPS! In: TEIJE, A. t.; VÖLKER, J.; HANDSCHUH, S.; STUCKENSCHMIDT, H.; D'ACQUIN, M.; NIKOLOV, A.; AUSSENAC-GILLES, N.; HERNANDEZ, N. (Ed.). **Knowledge Engineering and Knowledge Management**. Springer Berlin Heidelberg, 2012, (Lecture Notes in Computer Science, 7603). p. 267–281. ISBN 978-3-642-33875-5 978-3-642-33876-2. DOI: 10.1007/978-3-642-33876-2_24. Disponível em: <http://link.springer.com/chapter/10.1007/978-3-642-33876-2_24>. Citado na página 53.

PRUD'HOMMEAUX, E.; SEABORNE, A. **SPARQL Query Language for RDF**. [S.l.], 2008. Disponível em: <<http://www.w3.org/TR/rdf-sparql-query/>>. Citado nas páginas 46 e 47.

- RAKIB, A.; FARUQUI, R. U.; MACCAULL, W. Verifying Resource Requirements for Ontology-Driven Rule-Based Agents. In: LUKASIEWICZ, T.; SALI, A. (Ed.). **Foundations of Information and Knowledge Systems: 7th International Symposium, FoIKS 2012, Kiel, Germany, March 5-9, 2012. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 312–331. ISBN 978-3-642-28472-4. Disponível em: <http://dx.doi.org/10.1007/978-3-642-28472-4_18>. Citado na página 46.
- RECTOR, A.; DRUMMOND, N.; HORRIDGE, M.; ROGERS, J.; KNUBLAUCH, H.; STEVENS, R.; WANG, H.; WROE, C. OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns. In: **Engineering Knowledge in the Age of the Semantic Web**. Springer, Berlin, Heidelberg, 2004. p. 63–81. DOI: 10.1007/978-3-540-30202-5_5. Disponível em: <https://link.springer.com/chapter/10.1007/978-3-540-30202-5_5>. Citado na página 57.
- ROUSSEY, C.; CORCHO, O.; VILCHES-BLÁZQUEZ, L. M. A Catalogue of OWL Ontology Antipatterns. In: **Proceedings of the Fifth International Conference on Knowledge Capture**. New York, NY, USA: ACM, 2009. (K-CAP '09), p. 205–206. ISBN 978-1-60558-658-8. Disponível em: <<http://doi.acm.org/10.1145/1597735.1597784>>. Citado na página 57.
- SCHÖBER, D.; TUDOSE, I.; SVATEK, V.; BOEKER, M. OntoCheck: verifying ontology naming conventions and metadata completeness in Protégé 4. **Journal of Biomedical Semantics**, v. 3 Suppl 2, p. S4, set. 2012. ISSN 2041-1480. Citado nas páginas 56 e 58.
- SEIDENBERG, J.; RECTOR, A. Web Ontology Segmentation: Analysis, Classification and Use. In: **Proceedings of the 15th International Conference on World Wide Web**. New York, NY, USA: ACM, 2006. (WWW '06), p. 13–22. ISBN 1-59593-323-9. Disponível em: <<http://doi.acm.org/10.1145/1135777.1135785>>. Citado nas páginas 30 e 49.
- SHNEIDERMAN, B. Tree Visualization with Tree-maps: 2-d Space-filling Approach. **ACM Trans. Graph.**, v. 11, n. 1, p. 92–99, jan. 1992. ISSN 0730-0301. Disponível em: <<http://doi.acm.org/10.1145/102377.115768>>. Citado na página 76.
- SMITH, M. K.; WELTY, C.; MCGUINNESS, D. L. **OWL Web ontology language guide**. [S.l.], 2004. Disponível em: <<http://www.w3.org/TR/owl-guide/>>. Citado nas páginas 40, 41 e 42.
- STAAB, S.; MAEDCHE, A.; HANDSCHUH, S. **An annotation framework for the semantic web**. [s.n.], 2001. Disponível em: <<http://citeseer.ist.psu.edu/staab01annotation.html>>. Citado na página 38.
- STUDER, R.; BENJAMINS, V.; FENSEL, D. Knowledge engineering: Principles and methods. **Data & Knowledge Engineering**, v. 25, n. 1-2, p. 161–197, mar. 1998. ISSN 0169023X. Disponível em: <<http://linkinghub.elsevier.com/retrieve/pii/S0169023X97000566>>. Citado nas páginas 29 e 36.
- SUAREZ-FIGUEROA, M. C.; KAMEL, M.; POVEDA-VILLALON, M. Benefits of Natural Language Techniques in Ontology Evaluation : the OOPS! Case (short paper). In: CEA, G. Aguado de; AUSSÉNAC-GILLES, N. (Ed.). **Conférence Internationale sur la Terminologie et l'Intelligence Artificielle (TIA), Paris (F)**,

28/10/2013-30/10/2013. <http://www-lipn.univ-paris13.fr/>: LIPN - Université Paris 13, 2013. p. 5. Disponível em: <http://lipn.univ-paris13.fr/tia2013/Proceedings/actesTIA2013.pdf>-<http://oatao.univ-toulouse.fr/13198/>>. Citado na página 53.

SUGUMARAN, V.; GULLA, J. **Applied Semantic Web Technologies**. [S.l.]: Taylor & Francis, 2011. ISBN 978-1-4398-0156-7. Citado nas páginas 35 e 37.

SUN, Y.; ZHANG, J.; BIE, R.; WANG, H. Managing Rules in Semantic Web: Redundancy Elimination and Consistency Check. v. 5, n. 2, p. 191–200, fev. 2011. ISSN 1975-9339. Citado nas páginas 55, 58 e 69.

SURE, Y.; ANGELE, J.; STAAB, S. OntoEdit: Guiding Ontology Development by Methodology and Inferencing. In: MEERSMAN, R.; TARI, Z. (Ed.). **On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE: Confederated International Conferences CoopIS, DOA, and ODBASE 2002 Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. p. 1205–1222. ISBN 978-3-540-36124-4. DOI: 10.1007/3-540-36124-3_76. Disponível em: http://dx.doi.org/10.1007/3-540-36124-3_76>. Citado na página 75.

VILLALON, M. P.; SUÁREZ-FIGUEROA, M. C.; GÓMEZ-PÉREZ, A. A Double Classification of Common Pitfalls in Ontologies. In: **Proceedings of Workshop on Ontology Quality (OntoQual 2010), Co-located with EKAW 2010**. [s.n.], 2010. Disponível em: <http://oa.upm.es/5413/>>. Citado na página 57.

WANG, X. H.; ZHANG, D. Q.; GU, T.; PUNG, H. K. Ontology Based Context Modeling and Reasoning Using OWL. In: **Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops**. Washington, DC, USA: IEEE Computer Society, 2004. (PERCOMW '04), p. 18–22. ISBN 978-0-7695-2106-0. Disponível em: <http://dl.acm.org/citation.cfm?id=977405.978618>>. Citado na página 54.

YOUNG, L.; TU, S.; TENNAKOON, L.; VISMER, D.; ASTAKHOV, V.; GUPTA, A.; GRETHE, J.; MARTONE, M.; DAS, A.; MCAULIFFE, M. Ontology driven data integration for autism research. In: **22nd IEEE International Symposium on Computer-Based Medical Systems, 2009. CBMS 2009**. [S.l.: s.n.], 2009. p. 1–7. Citado nas páginas 17, 109, 123, 125, 134 e 135.

ZHANG, D. Fixpoint semantics for rule-base anomalies. In: **Fourth IEEE Conference on Cognitive Informatics, 2005. (ICCI 2005)**. [S.l.: s.n.], 2005. p. 10–17. Citado na página 49.