Pós-Graduação em Ciência da Computação

Yuri de Almeida Malheiros Barbosa

UM MÉTODO DE EXPANSÃO DE ONTOLOGIAS BASEADO EM QUESTÕES DE COMPETÊNCIA COM RASTREABILIDADE AUTOMÁTICA

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE
2017

Yuri de Almeida Malheiros Barbosa

UM MÉTODO DE EXPANSÃO DE ONTOLOGIAS BASEADO EM QUESTÕES DE COMPETÊNCIA COM RASTREABILIDADE AUTOMÁTICA

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Doutor em Ciência da Computação.

Orientador: *Frederico Luiz Gonçalves de Freitas*

RECIFE
2017

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

B238m Barbosa, Yuri de Almeida Malheiros
Um método de expansão de ontologias baseado em questões de competência com rastreabilidade automática / Yuri de Almeida Malheiros Barbosa. – 2017.
144 f.: il., fig., tab.

Orientador: Frederico Luiz Gonçalves de Freitas.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2017.
Inclui referências e apêndices.

1. Inteligência artificial. 2. Ontologia. I. Freitas, Frederico Luiz Gonçalves de (orientador). II. Título.

006.3 CDD (23. ed.) UFPE- MEI 2017-124

Yuri de Almeida Malheiros Barbosa

Um Método de Expansão de Ontologias Baseado em Questões de Competência com Rastreabilidade Automática

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutora em Ciência da Computação

Aprovado em: 23/02/2016.

_____

Orientador: Prof. Dr. Frederico Luiz Gonçalves de Freitas

BANCA EXAMINADORA

_____

Prof. Dr. Ruy José Guerra Barretto de Queiroz
Centro de Informática / UFPE

_____

Prof. Dr. Jaelson Freire Brelaz de Castro
Centro de Informática / UFPE

_____

Profa. Dra. Renata Wassermann
Instituto de Matemática e Estatística/USP

_____

Profa. Dra. Natasha Correia Queiroz Lino
Departamento de Informática/UFPB

_____

Profa. Dra. Renata Vieira
Faculdade de Informática/ PUC-RS

Eu dedico este trabalho aos meus pais, irmãos, minha namorada Graziele, professores, amigos e colegas que me ajudaram e me apoiaram durante toda a jornada para chegar até aqui.

Agradecimentos

O doutorado é um processo longo que me acompanhou em diversos momentos da minha vida e que me influenciou profundamente assim como as pessoas próximas a mim.

Eu gostaria de agradecer primeiramente a minha família, sem eles nada disso seria possível. Meus pais João Batista e Francinete que desde a minha infância me forneceram a melhor educação que eles poderiam me dar. Além de toda ajuda, incentivo e amor que um pai e uma mãe poderia ter. Também tenho que agradecer aos meus irmãos Dimitri e Igor pela amizade e por todos os momentos juntos que ajudaram a formar as pessoas que somos hoje.

Minha namorada Graziele, minha grande companheira e apoiadora. Sempre me colocando para cima nos momentos mais difíceis, de dúvidas e incertezas, e sempre me acompanhando também nos momentos alegres e de vitória. Um parágrafo é pouco para agradecer tudo o que você fez e faz por mim. Te amo.

Os amigos também foram muito importantes ao longo do meu doutorado. Todas as conversas, ideias trocadas e momentos compartilhados, me ajudaram a seguir em frente. Em especial queria agradecer aos amigos Adriano, Amanda, Bruno, Eduardo, Roger e Thiago.

Gostaria de agradecer aos meus colegas de trabalho que ao longo dos anos viraram amigos também. Muitos de vocês fizeram ou estão fazendo doutorado, assim as conversas e apoio mútuo foram muito importantes para todos nós. Mais especificamente, eu gostaria de agradecer aos amigos/colegas Ayla, Hacks, Jorge, Pasqueline, Rafael, Rodrigo e Scaico.

Meus colegas do grupo de pesquisa Sword, Adriano, Camila, Cleyton, Filipe e Ryan, que compartilharam comigo conversas e ideias e que ajudaram direta e indiretamente minha pesquisa. Saibam que tem um pouco de vocês no trabalho aqui apresentado.

Durante o doutorado eu passei quase um ano na Ryerson University em Toronto no Canadá como pesquisador visitante no laboratório LS3. Esse período foi de grande crescimento profissional e pessoal. Eu sou muito grato ao meu orientador na Ryerson University, Ebrahim Bagheri, um exemplo de pessoa e profissional que influenciou diretamente em quem sou hoje. Também agradeço aos meus colegas do LS3, Andisheh, Asef, Faezeh, Hossein, Luna, Mahdi, Mehrnaz, Mona, Thuan, Samane e Xin. Foram muitos almoços juntos, brincadeiras, discussões e trocas de experiências pessoais e profissionais. Espero ver vocês no futuro.

Agradeço a todos os professores do Centro de Informática da Universidade Federal de Pernambuco que contribuiram com minha formação e com o desenvolvimento desse trabalho.

Para finalizar, um agradecimento especial para o meu orientador Fred Freitas. O doutorado não é um processo fácil, mas sua orientação e seu apoio foram fundamentais na condução da minha pesquisa. Hoje tenho em você um modelo de pesquisador e mais importante um modelo de pessoa que trata muito bem os orientandos e consegue conduzir um trabalho complexo de forma tranquila. Muito obrigado por me ajudar a chegar até aqui.

It is paradoxical, yet true, to say, that the more we know, the more ignorant we become in the absolute sense, for it is only through enlightenment that we become conscious of our limitations. Precisely one of the most gratifying results of intellectual evolution is the continuous opening up of new and greater prospects.

—NIKOLA TESLA

Resumo

Questões de competência possuem um papel importante no ciclo de desenvolvimento de ontologias. Elas são amplamente usadas para definir requisitos de ontologias, entretanto a maioria das metodologias de desenvolvimento não especifica como a checagem dos requisitos deve ser feita ou apenas sugere que as questões sejam verificadas manualmente. Assim, faltam ferramentas que suportem checagem automática, inclusive utilizando raciocinadores, que podem agilizar o desenvolvimento e melhorar a qualidade das ontologias produzidas. Além disso, a rastreabilidade de requisitos raramente é explorada, mesmo essa atividade sendo estudada e utilizada por anos na engenharia de software. Nesse trabalho são introduzidos um método iterativo para expansão de ontologias usando questões de competência e uma ferramenta que implementa o método. Várias inovações são apresentadas: um componente que traduz questões de competência em linguagem natural para lógica de descrição para efetuar checagem automática usando raciocinadores; um gerador de questões de competência para guiar engenheiros durante o desenvolvimento; um componente que escreve código OWL de acordo com perguntas e respostas; um rastreador que monitora as relações entre requisitos e código e vice-versa; e um método que integra todos os pontos anteriores, criando uma maneira iterativa de expandir ontologias através de perguntas e respostas semelhante a um diálogo controlado. Para avaliar o método proposto e sua implementação foram executados testes com a ontologia SNOMED CT para analisar o comportamento dos componentes criados. Também foram realizados dois estudos de caso para avaliar o uso da ferramenta por usuários. Os testes mostraram a capacidade do método em checar e adicionar conhecimento a uma ontologia. Foi possível criar perguntas para checar todos os axiomas escolhidos da SNOMED CT e a implementação conseguiu sugerir perguntas para adicionar conhecimento à ontologia em 69,1% dos casos. Os estudos de caso levantaram os pontos fortes e fracos da implementação, mostrando o potencial da implementação em melhorar o desenvolvimento de ontologias, pois a interação através de linguagem natural é simples tanto para checagem quanto para adição de axiomas, mesmo para usuários leigos, e a rastreabilidade de requisitos grava e apresenta informações importantes para o engenheiro de ontologias.

Palavras-chave: Ontologias. Engenharia de ontologias. Questões de competência. Lógicas de descrição.

Abstract

Competency questions have an important role in the development of ontologies. Usually, they are used as ontology requirements, however many ontology development methodologies do not detail how to check the requirements or only suggest checking the questions manually. Thus, there is a lack of tools to check competency questions automatically, including using reasoners, that could make the development faster and could improve the quality of ontologies. Furthermore, requirement traceability for ontology engineering is rarely explored, even though it is studied and used by software engineers for years. In this work, we introduce an iterative method to expand ontologies using competency questions, and a tool that implements this method. Many novel approaches are presented: a component that translates natural language competency questions to description logics to check them automatically using reasoners; a component that generates competency questions to guide engineers; a component that writes OWL code using questions and answers; a tracker that monitors the relations among requirements and code and vice-versa; and a method that integrates all previous components to create an iterative way to expand ontologies using questions and answers similar to a controlled dialogue. To evaluate the method and its implementation we ran tests using the SNOMED CT ontology to analyze the behavior of the developed components. Also, we did two case studies, thus users could evaluate the tool. The tests showed the capacity of the method to check and add knowledge to an ontology. We could create questions to check all chosen axioms of SNOMED CT, and the implementation was able to suggest questions to add knowledge in 69,1% of cases. The case studies exposed the strength and weakness of the implementation. They showed the implementation potential to improve the ontology development, because it is simple to interact using natural language to check and to add axioms, even for non-experts users. Also, the requirement traceability stores and presents important information for the ontology engineers.

Keywords: Ontology. Ontology engineering. Competency questions. Description logics.

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

DL	Lógica de Descrição
IHTSDO	The International Health Terminology Standards Development Organisation
OTKM	On-To-Knowledge Methodology
OWL	Web Ontology Language
QC	Questão de Competência
QS	Questão do Sistema
TDD	Test Driven Development
W3C	World Wide Web Consortium

Sumário

1 Introdução

1.1 Motivação

Por muitos anos, o desenvolvimento de ontologias era realizado através de esforços ad hoc, ou seja, cada equipe adotava seu próprio conjunto de princípios, critérios e fases de projeto para construção desse tipo de artefato. Isso distanciou inicialmente os esforços da engenharia de ontologias do rigor de outros ramos mais conhecidos da engenharia (Guarino & Welty, 2002) (Gómez-Pérez *et al.* , 2004). Portanto, os engenheiros de ontologias não possuíam um compêndio para guiar a construção de ontologias, mostrando atividades, ciclo de vida, técnicas e ferramentas.

Em um sinal claro de progresso, metodologias consistentes e diversas ferramentas vêm sendo propostas para apoiar o desenvolvimento de ontologias. As metodologias abordam atividades de criação e manutenção, especificam o ciclo de vida e definem, por exemplo, como descrever o escopo e os requisitos, como criar a especificação de uma ontologia e como conduzir sua evolução. Os requisitos frequentemente são definidos como perguntas que devem ser respondidas usando o conhecimento codificado na ontologia chamadas de questões de competência (QCs) (Gruninger & Fox, 1995). Tais esforços organizaram e definiram atividades padronizadas para guiar os engenheiros durante todo o processo de construção de ontologias.

Entre as metodologias de desenvolvimento de ontologias mais conhecidas estão: Methontology (Fernandez-Lopez *et al.* , 1997), On-To-Knowledge (Staab *et al.* , 2001), Ontology 101 (Noy & McGuinness, 2001) e NeOn (del Carmen Suárez-Figueroa *et al.* , 2008). É importante salientar, que apesar de diferentes, essas metodologias possuem muitas características em comum. Duas das mais importantes consistem na forma iterativa de desenvolvimento e no uso de QCs para definição de requisitos. Existem também várias ferramentas para auxiliar o desenvolvimento de ontologias. Protégé (Gennari *et al.* , 2003), OntoStudio[1], NeOn Toolkit (del Carmen Suárez-Figueroa *et al.* , 2008), OntoEdit (Sure *et al.* , 2002) e WebODE (Arpírez *et al.* , 2001) estão entre as ferramentas mais usadas para facilitar o processo de construção de uma ontologia.

Apesar dos avanços na engenharia de ontologias, ainda existem muitas áreas que podem ser melhoradas. Por exemplo, as QCs são usadas amplamente para definir requisitos, entretanto

[1] http://www.semafora-systems.com/en/products/ontostudio/

a maioria das metodologias não especifica como a checagem deve ser feita ou sugere que as QCs sejam verificadas manualmente. A checagem manual cria um gargalo no processo de desenvolvimento, pois avaliar um grande número de questões é uma tarefa difícil, propícia a erros e que pode levar muito tempo para ser executada. Convertendo as QCs para consultas SPARQL (Prud'Hommeaux *et al.* , 2008) consegue-se um grau de automatização para evitar o gargalo mencionado. Por outro lado, consultas SPARQL não suportam inferência de forma independente, para isso é necessário executar um raciocinador automático antes de efetuar as consultas. Assim, outra forma de checar conhecimento em uma ontologia é usar um raciocinador para fazer consultas em lógicas de descrição (DL) diretamente, as quais já consideram os conhecimentos inferidos. Automatizar o processo de checagem de QCs pode agilizar o desenvolvimento e melhorar a qualidade das ontologias produzidas, pois a checagem automatizada é mais rápida que a manual. Além disso, verificar os requisitos tendo a certeza que todos eles estão sendo satisfeitos faz os artefatos produzidos atenderem as necessidades dos clientes que os usarão.

A rastreabilidade de requisitos, ou seja, a habilidade de descrever e seguir o ciclo de vida dos requisitos (Gotel & Finkelstein, 1994), é usada na engenharia de software há décadas (Winkler & Pilgrim, 2010). Entretanto, ela é uma área pouco explorada na engenharia de ontologias, que poderia trazer ganhos significativos para o processo de desenvolvimento. As ontologias especificadas em DL têm uma base lógica formal que possibilita rastrear mudanças automaticamente de uma maneira difícil de replicar na engenharia de software. Por exemplo, é possível:

- Checar quais requisitos estão associados a determinados axiomas e vice-versa;
- Checar automaticamente, inclusive com inferência através de raciocinadores, se todos os requisitos estão sendo satisfeitos;
- Avaliar a cada mudança na ontologia se todos os requisitos continuam sendo satisfeitos.

Por fim, criar axiomas não é uma atividade simples. Modelar um domínio através de uma ontologia não é trivial e exige o conhecimento de diversos conceitos para que a ontologia seja criada de forma correta. Além disso, muitos especialistas em um certo domínio podem não saber criar ontologias, assim dificultando a colaboração dessas pessoas no processo de desenvolvimento. Por isso, usar uma interface mais simples, como a interação através de linguagem natural, pode facilitar a inclusão de não especialistas na construção de uma ontologia.

Os problemas levantados anteriormente são o foco do trabalho apresentado. Nessa tese foi explorado e melhorado o uso de QCs através de checagens automáticas e de geração de perguntas para guiar o engenheiro no desenvolvimento de ontologias. Além disso, foi criado um sistema de rastreabilidade que interliga automaticamente os requisitos ao código da ontologia e provê operações para que o engenheiro analise e manipule o histórico de desenvolvimento da ontologia. A automatização da criação de ligações entre código e requisitos não existe

completamente na engenharia de software, nem foi proposta anteriormente para engenharia de ontologias.

1.2 Objetivos

O objetivo geral deste trabalho é criar um método, com uma ferramenta de suporte, de expansão de ontologias, ou seja, adicionar conhecimento a uma ontologia já existente, focado em questões de competência que possua um sistema de rastreabilidade automática de requisitos.

Como objetivos específicos, temos:

- Definir um método de expansão de ontologias focado em questões de competência;
- Construir um sistema de checagem automática de ontologias usando QCs em linguagem natural controlada;
- Implementar um gerador automático de QCs;
- Implementar um sistema para expansão de ontologias usando o método definido;
- Criar um sistema de rastreabilidade automática de requisitos.

1.3 Visão Geral da Solução Proposta

Neste trabalho, são propostos um método e uma ferramenta que implementa o método de expansão de ontologias. Eles aperfeiçoam abordagens já conhecidas na engenharia de ontologias, como o uso de QCs, e introduzem outras pouco exploradas, como a rastreabilidade de requisitos. Usando o método proposto, um engenheiro pode expandir ontologias iterativamente usando QCs e as respostas dessas perguntas.

Como em outros trabalhos, as QCs são usadas como requisitos, mas o método vai além, ele usa QCs escritas em linguagem natural para checar automaticamente ontologias escritas em OWL DL (McGuinness & Van Harmelen, 2004). Dessa forma, um engenheiro utilizando o método não precisa conhecer a sintaxe da DL, pois toda a interação é feita através de linguagem natural, diminuindo assim as barreiras para não especialistas em DL. Cada QC corresponde a um axioma em DL que deve estar presente na ontologia ou deve ser inferido através de um raciocinador. Isso possibilita a checagem automática de QCs em linguagem natural, diferindo de trabalhos na área que usualmente checam as ontologias de forma manual.

Usando uma ontologia classificada é possível consultar usando SPARQL os conhecimentos inferidos por um raciocinador, entretanto Rector (2003) mostra que usar raciocinadores na construção de ontologias facilita a modularização, reuso e manutenção. Como tem-se um método para desenvolver ontologias, a abordagem de raciocínio em DL foi escolhida, ao invés da consulta de ontologias classificadas usando SPARQL.

No método, as interações do usuário durante o processo de desenvolvimento de uma ontologia são gravadas. Mais detalhadamente são gravadas as QCs perguntadas, as respostas das QCs, os axiomas que elas representam, os conceitos e axiomas que elas estão relacionadas e o código OWL modificado durante o processo. Assim, um engenheiro tem acesso a diversos rastros deixados durante o desenvolvimento para que ele possa analisar o histórico das atividades realizadas, como os requisitos se relacionam, qual o impacto deles nos axiomas e o impacto dos axiomas nos requisitos.

Por fim, a natureza iterativa do método se encaixa bem em diversos processos descritos na literatura, mostrando o alinhamento dele com os avanços na área de engenharia de ontologias. Portanto, o trabalho aqui apresentado pode ser usado para melhorar processos já conhecidos e utilizados por outros engenheiros, sem que uma quebra com os paradigmas atuais seja efetuada.

O uso da ferramenta que implementa o método segue os passos apresentados no fluxograma que pode ser visto na Figura 1.1.

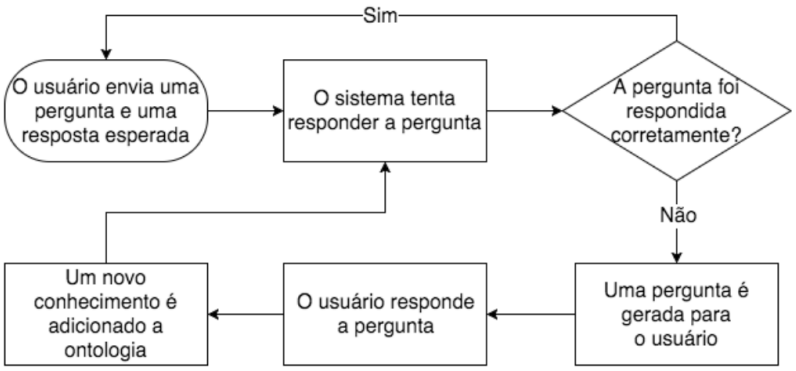

Figura 1.1: Fluxograma do funcionamento do método de expansão de ontologias

É importante perceber que além de conseguir checar axiomas em DL usando inferência, o método possui uma etapa na qual novas QCs são geradas automaticamente quando uma QC inicial não é respondida corretamente. Para diferenciar as perguntas geradas das questões de competência criadas pelos usuários, chamaremos as questões geradas automaticamente de questões do sistema (QSs). O objetivo das QSs é guiar o engenheiro de ontologias na adição de novos axiomas para que uma QC passe a ser respondida corretamente. Ao responder uma QS, novos conhecimentos são adicionados à ontologia. Assim, o método, através de perguntas e respostas, permite um engenheiro checar se requisitos são satisfeitos e se não forem, ele sugere axiomas através das QSs para satisfazê-los.

1.4 Contribuições Realizadas

Como resultado do trabalho apresentado, pode-se destacar as seguintes contribuições:

- A criação de um método para expansão iterativa e interativa de ontologias usando questões de competência;
- A implementação de uma ferramenta para expandir ontologias em DL $\mathcal{EL}$ utilizando o método proposto;
- A implementação de um sistema de checagem automática de questões de competência escritas em linguagem natural controlada;
- O uso de unificação em DL $\mathcal{EL}$ para realizar abdução em TBox;
- A implementação de um sistema de rastreabilidade automática de requisitos que foi incorporado a ferramenta desenvolvida;
- A disponibilização da ferramenta desenvolvida [2].

Além das contribuições mencionadas, durante o desenvolvimento deste trabalho foram produzidos os artigos:

- MALHEIROS, Y; FREITAS, F. A Method to Develop Description Logic Ontologies Iteratively Based on Competency Questions: an Implementation. In: ONTOBRAS, 2013, Belo Horizonte. Proceedings of the 6th Seminar on Ontology Research in Brazil, 2013. p. 142-153.
- MALHEIROS, Y ; FREITAS, F. A Method to Develop Description Logic Ontologies Iteratively with Automatic Requirement Traceability. In: 27th International Workshop on Description Logics, 2014, Viena. Proceedings of the 27th International Workshop on Description Logics, 2014. p. 646-658.

1.5 Organização da Tese

Além do capítulo de introdução, este trabalho envolve outros seis capítulos, que estão organizados da seguinte forma:

Capítulo 2: Apresenta o referencial teórico do trabalho. Nele são definidos e explicados conceitos sobre ontologias, web semântica, *Web Ontology Language* (OWL), lógica de descrição, questões de competência, engenharia de ontologias e rastreabilidade de requisitos;

Capítulo 3: Apresenta e discute diversas metodologias para engenharia de ontologias;

Capítulo 4: Descreve e explica o método proposto, como ele funciona e quais são suas etapas;

[2] A ferramenta pode ser encontrada no endereço: https://yurimalheiros.github.io/ionscq/

Capítulo 5: Detalha a implementação e o funcionamento da ferramenta para expansão iterativa de ontologias utilizando o método proposto. Neste capítulo também são explicados o funcionamento do sistema de rastreabilidade automática e do mecanismo de geração de perguntas.

Capítulo 6: Apresenta os experimentos e os estudos de caso realizados para avaliar o método de expansão de ontologias e sua implementação;

Capítulo 7: Traz as conclusões do trabalho enfatizando as contribuições realizadas, limitações do trabalho e direcionamentos para trabalhos futuros.

2

Referencial Teórico

Neste capítulo são apresentados os conceitos que servem como base para o entendimento do restante do trabalho. Nas seções seguintes são explicados o que são ontologias, lógica de descrição, web semântica, *Web Ontology Language* (OWL), questões de competência, engenharia de ontologias e rastreabilidade de requisitos.

2.1 Ontologias

Na literatura existem diferentes definições para ontologias, entretanto uma das definições mais aceitas foi escrita por Gruber (1995): uma ontologia é uma especificação explícita de uma conceitualização. Nesse contexto, conceitualização significa um modelo sobre algum domínio de conhecimento na forma de definições de conceitos, propriedades e as relações entre eles. Já especificação explícita significa que o modelo deve ser especificado em uma linguagem não ambígua, que seja acessível para máquinas e também para humanos.

As ontologias podem ser usadas em diferentes tipos de aplicações. De acordo com Uschold & Gruninger (1996) as ontologias podem solucionar ou amenizar problemas relacionados à comunicação entre pessoas, organizações e sistemas de software, que devem ser resolvidos eliminando ou reduzindo a falta de conhecimento sobre conceitos envolvidos nos processos de comunicação. Por exemplo, pesquisadores de diferentes áreas podem ter dificuldade em usar os avanços dos trabalhos uns dos outros, pois, apesar de existirem ideias semelhantes, eles podem usar termos diferentes para descrevê-las, assim prejudicando o entendimento. Uma solução seria identificar ideias em comum entre diferentes áreas do conhecimento e os termos relacionados à elas. Em seguida, após uma análise rigorosa, interligar os conceitos análogos para que seja possível fazer uma tradução de uma área para outra. Com isso, pesquisadores conseguiriam ter um melhor entendimento de trabalhos publicados fora de sua área de atuação.

Já em um contexto de desenvolvimento de sistemas de software, tem-se os seguintes possíveis problemas que podem ser resolvidos através do uso de ontologias:

- Comunicação ruim entre pessoas e organizações;

- Dificuldade de identificar requisitos e de especificar um sistema de software;
- Interoperabilidade entre paradigmas, linguagens e ferramentas;
- Reuso de paradigmas, linguagens e ferramentas.

Além de servir para remover ambiguidades na interpretação de conceitos, as ontologias também costumam ser usadas como base de conhecimento. Nelas temos uma rede de conceitos interligados através de suas relações que podem ser explorados por sistemas de software que proveem acesso às informações da base de conhecimento. Essas ontologias podem tirar proveito do uso de raciocinadores automáticos para inferir mais relações, gerando mais conhecimento.

Existem diferentes formas de se criar ontologias, entretanto é comum que elas trabalhem com as ideias a seguir:

- Classes (ou conceitos): representam classes de elementos com características semelhantes;
- Indivíduos: são instâncias de uma classe;
- Relações (ou papéis): relacionam elementos como classes e indivíduos;
- Axiomas: definições que formam o conhecimento representado pela ontologia.

Uma visualização gráfica de uma ontologia é apresentada na Figura 2.1. Ela mostra as principais classes e papéis de uma ontologia sobre manufatura apresentada no trabalho de Lemaignan *et al.* (2006). Nela pode-se perceber a relação hierárquica entre classes, por exemplo, tem-se que um *Raw material* é um tipo de *Techonological entity*. Além disso, tem-se papéis que relacionam classes, como *executes* que relaciona *Human resource* e *Operation*.

Nas próximas subseções são apresentados conceitos de lógica de descrição, web semântica e OWL. Os três tópicos juntos formam a base das tecnologias usadas nessa pesquisa para se trabalhar com ontologias.

2.1.1 Lógicas de Descrição

Lógicas de Descrição (DL) (Baader, 2003) são uma família de formalismos de representação de conhecimento usados para representar e raciocinar sobre elementos de um determinado domínio de aplicação. As DLs tem ganhado um interesse crescente nas últimas duas décadas, principalmente após a aprovação da Web Ontology Language (OWL) (Patel-Schneider *et al.* , 2004), que tem DL como base, como padrão da *World Wide Web Consortium* (W3C).

Para definir uma base de conhecimento, a DL utiliza três elementos básicos conhecidos como conceitos, papéis e indivíduos, que podem ser combinados através de um conjunto de construtores.

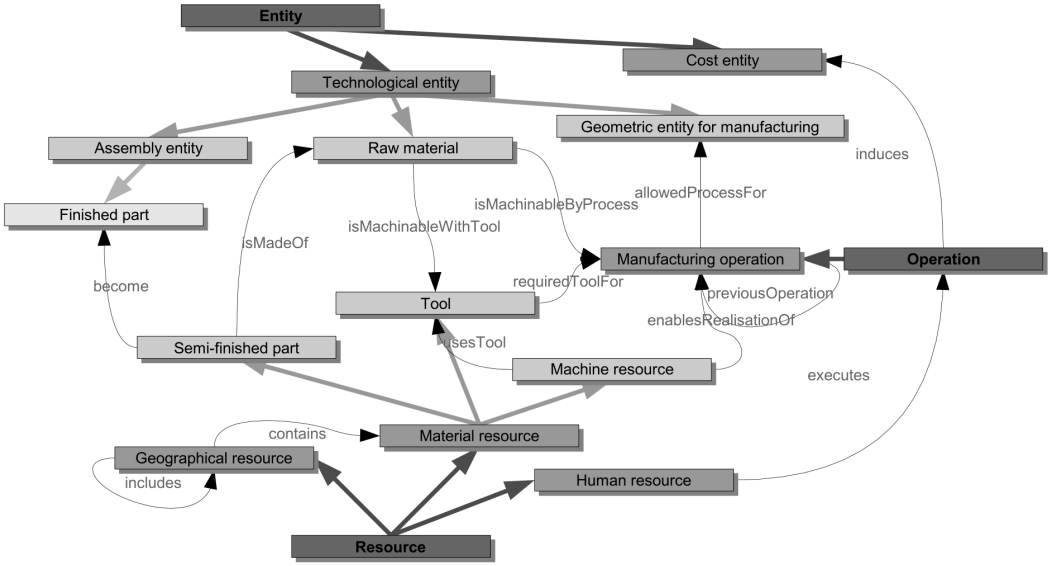

Figura 2.1: Principais classes e papeis de um ontologia sobre manufatura

Fonte: (Lemaignan *et al.* , 2006)

Definição 2.1.1. (Conceitos) Um conceito é utilizado para representar uma classe de elementos que compartilham características em comum. Por exemplo, o conceito *Car* pode ser usado para todos os veículos desse tipo ou o conceito *Woman* pode ser usado para representar todas as pessoas do sexo feminino.

Definição 2.1.2. (Papéis) Um papel é utilizado para representar uma relação binária entre dois elementos que podem ser conceitos, indivíduos ou valores (números, *strings*, etc.). Por exemplo, um papel *ownCar* pode relacionar os conceitos *Woman* e *Car*, especificando que uma mulher é dona de um carro.

Definição 2.1.3. (Indivíduos) Indivíduos são instâncias de uma determinada classe (conceito). Por exemplo, Maria pode ser uma instância do conceito *Woman*, ou Fusca uma instância do conceito *Car*.

A DL possui semântica baseada em lógica de primeira ordem que é definida através de interpretações. Dado que N_c, N_r e N_i são conjuntos não vazios de conceitos, papéis e indivíduos respectivamente. Uma interpretação $\mathcal{I}$ consiste em um domínio $\Delta^{\mathcal{I}}$ e uma função de interpretação $.^{\mathcal{I}}$, formando o par $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$. $\Delta^{\mathcal{I}}$ é um conjunto não vazio e $.^{\mathcal{I}}$ é uma função que mapeia cada conceito $A \in N_c$ para um subconjunto de $\Delta^{\mathcal{I}}$, cada papel $R \in N_r$ para um subconjunto de $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ e cada indivíduo $a \in N_i$ para um elemento de $\Delta^{\mathcal{I}}$.

Uma base de conhecimento em DL é composta por duas partes principais: a TBox (*terminological box*) e a ABox (*assertional box*). A seguir tem-se a definição formal de ambas.

Definição 2.1.4. (TBox) Sejam C, D dois conceitos, uma TBox $\mathcal{T}$ é um conjunto finito, possivelmente vazio, de definições de inclusões de conceitos ($C \sqsubseteq D$) e de equivalência de conceitos

($C \equiv D$). A segunda é uma abreviação para $C \sqsubseteq D$ e $D \sqsubseteq C$. As definições em $\mathcal{T}$ são denominadas axiomas terminológicos. $\mathcal{T}$ é consistente se existe uma interpretação $\mathcal{I}$ que satisfaz $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ para cada $C \sqsubseteq D \in \mathcal{T}$ e que satisfaz $C^{\mathcal{I}} = D^{\mathcal{I}}$ para cada $C \equiv D \in \mathcal{T}$. Nesse caso, $\mathcal{I}$ é denominado modelo de $\mathcal{T}$.

Definição 2.1.5. (ABox) Sejam a, b dois indivíduos, C um conceito e R um papel, uma ABox $\mathcal{A}$ é um conjunto finito, possivelmente vazio, de asserções de conceitos ($a : C$), ou asserções de papéis ($\langle a, b\rangle : R$). As definições em $\mathcal{A}$ são denominadas asserções. $\mathcal{A}$ é consistente em relação a TBox $\mathcal{T}$ se existe um modelo $\mathcal{I}$ de $\mathcal{T}$ no qual $a^{\mathcal{I}} \in C^{\mathcal{I}}$ é satisfeito para cada $a : C$ em $\mathcal{A}$ e $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ é satisfeito para cada $\langle a, b\rangle : R \in \mathcal{A}$. Nesse caso, $\mathcal{I}$ é denominado modelo de $\mathcal{A}$.

Para concluir, após a definição dos seus componentes, tem-se a definição formal de base de conhecimento.

Definição 2.1.6. (Base de conhecimento) Uma base de conhecimento Σ é definida como uma tupla $\Sigma = (\mathcal{T}, \mathcal{A})$, onde $\mathcal{T}$ é uma TBox e $\mathcal{A}$ é uma ABox. Uma interpretação $\mathcal{I}$ satisfaz Σ se $\mathcal{I}$ for um modelo de $\mathcal{T}$ e se $\mathcal{I}$ for um modelo de $\mathcal{A}$.

Diferentes linguagens de lógica de descrição podem ser definidas de acordo com os construtores que elas suportam. No restante desse capítulo vamos usar as seguintes notações:

- A e B são conceitos atômicos;
- C e D são conceitos complexos;
- R e S são papéis.

A linguagem $\mathcal{AL}$ (Schmidt-Schauß & Smolka, 1991) é uma das variações mais simples da DL. Ela suporta os seguintes construtores:

- A (conceito atômico)
- $\top$ (conceito universal);
- $\bot$ (conceito base);
- $\neg A$ (complemento de um conceito atômico);
- $C \sqcap D$ (conjunção de dois conceitos);
- $\forall R.C$ (restrição universal de um conceito por um papel);
- $\exists R.\top$ (restrição existencial limitada).

A partir da linguagem $\mathcal{AL}$ pode-se adicionar novos construtores estendendo-a para gerar novas linguagens, formando o conjunto de linguagens da família $\mathcal{AL}$. Cada novo construtor é associado a uma letra, por exemplo, $\mathcal{C}$ é usada para o construtor de negação de conceitos complexos e $\mathcal{E}$ para o de restrição existencial. Através dessas letras, define-se a nomenclatura das linguagens de acordo com os construtores adicionados. Por exemplo, se adicionarmos a negação de conceitos complexos à linguagem $\mathcal{AL}$, tem-se agora a linguagem $\mathcal{ALC}$, já se adicionarmos o construtor de restrição existencial à $\mathcal{AL}$ tem-se a linguagem $\mathcal{ALE}$. A Tabela 2.1 apresenta diversos construtores que podem ser utilizados na DL, suas letras correspondentes, suas sintaxes e suas semânticas.

Tabela 2.1: Construtores da Lógica de Descrição

Construtor	Letra	Sintaxe	Semântica
Conceito universal	$\mathcal{AL}$	$\top$	$\Delta^{\mathcal{I}}$
Conceito base	$\mathcal{AL}$	$\bot$	$\emptyset$
Conjunção	$\mathcal{AL}$	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Negação	$\mathcal{C}$	$\neg A$	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
Disjunção	$\mathcal{U}$	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Restrição existencial	$\mathcal{E}$	$\exists R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b : (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$
Restrição universal	$\mathcal{AL}$	$\forall R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b : (a,b) \in R^{\mathcal{I}} \Rightarrow b \in C^{\mathcal{I}}\}$
Restrição numérica	$\mathcal{N}$	$\geqslant nR$	$\{a \in \Delta^{\mathcal{I}} \mid \mid \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\}\mid \geq n\}$
		$\leqslant nR$	$\{a \in \Delta^{\mathcal{I}} \mid \mid \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\}\mid \leq n\}$
		$= nR$	$\{a \in \Delta^{\mathcal{I}} \mid \mid \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\}\mid = n\}$
Restrição numérica qualificada	$\mathcal{Q}$	$\geqslant nR.C$	$\{a \in \Delta^{\mathcal{I}} \mid \mid \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}\mid \geq n\}$
		$\leqslant nR.C$	$\{a \in \Delta^{\mathcal{I}} \mid \mid \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}\mid \leq n\}$
		$= nR.C$	$\{a \in \Delta^{\mathcal{I}} \mid \mid \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}\mid = n\}$
Hierarquia de papéis	$\mathcal{H}$	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$

Nesse trabalho são utilizadas ontologias descritas na linguagem $\mathcal{EL}$, que também forma sua própria família. Os seus construtores são:

- $C \sqcap D$ (conjunção);
- $\exists R.C$ (restrição existencial);
- $\top$ (conceito universal).

Apesar de pouco expressiva, a $\mathcal{EL}$ é usada em grandes ontologias como a SNOMED[1] e a GALEN (Rogers & Rector, 1996). Além disso, o problema de inferência de subsunções em $\mathcal{EL}$ é polinomial, tornando o tempo de resposta usualmente rápido o suficiente para aplicações, mesmo com ontologias grandes.

A DL se torna ainda mais interessante, pois ela oferece serviços de inferência, que são utilizados para raciocinar sobre a base de conhecimento em DL e descobrir conhecimentos

[1] http://www.ihtsdo.org/snomed-ct/

implícitos que podem ser inferidos. Os serviços de inferência podem ser utilizados tanto na TBox quanto na ABox. A seguir são descritos os principais serviços de inferência da DL.

Satisfatibilidade de conceitos: checa se $\Sigma \not\models C \equiv \bot$, ou seja, um conceito C é dito como satisfatível em relação a uma base de conhecimento Σ, se e somente se existe um modelo $\mathcal{I}$ de Σ no qual $C^{\mathcal{I}} \neq \emptyset$.

Por exemplo, em uma ontologia com os axiomas:

- $Parent \equiv Person \sqcap \exists hasChild.Person$
- $Woman \equiv Female \sqcap Person$
- $Mother \equiv Female \sqcap Parent$

Vamos verificar se $\neg Woman \sqcap Mother$ é satisfatível. Para isso, expande-se o conceito seguindo os passos:

1. $\neg Woman \sqcap Mother$
2. $\neg(Female \sqcap Person) \sqcap Female \sqcap Parent$
3. $(\neg Female \sqcup \neg Person) \sqcap Female \sqcap Parent$
4. $\neg Female \sqcap Female \sqcap Parent$
5. $\neg Person \sqcap Female \sqcap Parent$
6. $\neg Person \sqcap Female \sqcap Person \sqcap \exists hasChild.Person$

Nos itens 4 e 6 temos contradições. No primeiro pode ser observado $\neg Female \sqcap Female$ e no segundo $\neg Person \sqcap Person$. Assim, o conceito$\neg Woman \sqcap Mother$ não é satisfatível.

Subsunção: checa se $\Sigma \models C \sqsubseteq D$, ou seja, D subsume C, se e somente se todos os modelos $\mathcal{I}$ de Σ satisfizerem $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.

Por exemplo, dada a TBox a seguir:

- $Woman \equiv Person \sqcap Female$
- $Man \equiv Person \sqcap \neg(Person \sqcap Female)$
- $Mother \equiv (Person \sqcap Female) \sqcap \exists hasChild.Person$
- $Father \equiv (Person \sqcap \neg(Person \sqcap Female)) \sqcap \exists hasChild.Person$
- $Wife \equiv (Person \sqcap Female) \sqcap \exists hasHusband.(Person \sqcap \neg(Person \sqcap Female))$

É possível usar a subsunção para verificar, por exemplo, que $Woman \sqsubseteq Person$ e $Woman \sqsubseteq Female$. Ou ainda que $Wife \sqsubseteq Woman$, assim como $Wife \sqsubseteq Person$ e $Wife \sqsubseteq Female$, entretanto $Wife$ não subsume $Mother$ ou vice-versa. Por fim, temos que Man e $Woman$ são disjuntos, isso acontece, pois Man é subsumido por $\neg(Person \sqcap Female)$, que é a negação da definição de $Woman$.

Consistência da base de conhecimento: checa se $\Sigma \not\models \bot$, ou seja, se existe um modelo $\mathcal{I}$ de Σ.

Uma das formas de verificar a consistência de uma base de conhecimento é utilizar a técnica conhecida como Tableaux (Buchheit *et al.* , 1993). Sua ideia básica é tentar construir um modelo de forma incremental a partir dos axiomas. Para isso são aplicadas regras de transformação à base de conhecimento até que nenhuma regra possa ser usada. Cada regra tem uma condição para ser utilizada e produz um efeito diferente. Se, durante o processo, forem encontradas contradições, então a base de conhecimento é inconsistente. Se não forem encontradas contradições, então a base é consistente.

Dado um conjunto A inicial com os axiomas da base de conhecimento, a cada aplicação de uma regra de transformação, um ou mais novos conjuntos são gerados. A próxima aplicação de uma regra deve ser feita utilizando os conjuntos gerados. A Tabela 2.2 apresenta as regras de transformação do Tableaux, especificando a condição para o uso de cada regra e o efeito da sua aplicação para gerar novos conjuntos.

Por exemplo, dada a ABox a seguir:

- $(Parent \sqcap \forall child.Male)(john)$
- $\neg Male(mary)$
- $child(john, mary)$

Aplicando Tableaux pra verificar sua consistência, temos:

- $(Parent \sqcap \forall child.Male)(john)$
- $\neg Male(mary)$
- $child(john, mary)$
- $Parent(john)$ (regra - $\sqcap$)
- $(\forall child.Male(john))$
- $Male(mary)$ (regra - $\forall$)

Tabela 2.2: Regras de transformação do Tableaux

Regra	Condição	Efeito
$regra - \sqcap$	A contém $((C_1 \sqcap C_2)(x))$, mas não $C_1(x)$ e $C_2(x)$	$A' = A \cup \{C_1(x), C_2(x)\}$
$regra - \sqcup$	A contém $((C_1 \sqcup C_2)(x))$, mas não $C_1(x)$ ou $C_2(x)$	$A' = A \cup \{C_1(x)\}, A'' = A \cup \{C_2(x)\}$
$regras - \exists$	A contém $(\exists r.C(x))$, mas não $C(z)$ e $r(x,z)$, sendo z um indivíduo	$A' = A \cup \{C(y), r(x,y)\}$
$regra - \forall$	A contém $(\forall r.C(x)$ e $r(x,y)$, mas não $C(y))$	$A' = A \cup \{C(y)\}$

No último passo foi gerada uma contradição, pois agora tem-se $\neg Male(mary)$ e $Male(mary)$. Assim, a base de conhecimento é inconsistente. Se o procedimento usando Tableaux terminar (não for mais possível aplicar regras) e nenhuma contradição for encontrada, então a base é consistente.

Verificação de instância: seja a um indivíduo, checa se $\Sigma \models a : C$, ou seja, se a é uma instância do conceito C. A checagem é positiva se $a^{\mathcal{I}} \in C^{\mathcal{I}}$ para todos os modelos $\mathcal{I}$ de Σ.

Para poder ser utilizado em aplicações, é importante que o processo de inferência em DL forneça respostas em um tempo não muito longo e que esse processo sempre termine, mesmo que a resposta seja negativa. A decidibilidade e complexidade do processo de raciocínio automático dependem da expressividade da linguagem escolhida. Uma linguagem muito expressiva consegue representar conceitos complexos, mas a complexidade para inferir novas relações pode ser grande e em alguns casos o processo de inferência é indecidível. Já as linguagens menos expressivas permitem inferências de forma eficiente, mas podem não ser suficientes para expressar alguns conceitos em um domínio de uma aplicação.

2.1.2 Web Semântica

A World Wide Web mudou a forma como pessoas publicam e consomem informações. Hoje é possível enviar os mais diversos tipos de dados através da rede tornando-os acessíveis para os usuários da Internet de todo o planeta. A maior parte do conteúdo da Web é apresentada em documentos para o consumo de seres humanos, eles podem ter textos, imagens, tabelas, formulários, vídeos, e muitos outros elementos para exibir informações ou para receber entrada de dados. Além disso, os documentos podem conter referências para outros documentos, os *hyperlinks*, que interligam o conteúdo da Web.

Entretanto, a Web de hoje ainda não consegue prover de forma eficiente conteúdo que seja facilmente interpretado por máquinas, para que além dos humanos, programas de computador também consigam aproveitar o potencial da quantidade imensa de informações na Web. A evolução da representação de conteúdo da Web atual, para que ele seja processado por máquina de forma mais simples, resultará na Web Semântica, também conhecida como a Web de Dados. Nessa rede, os dados devem ser acessados usando a arquitetura existente da Web e eles devem poder ser ligados assim como os documentos são ligados através de *hyperlinks* (Berners-Lee *et al.* , 2001).

A Web Semântica pode ser utilizada para diversos tipos de aplicações que exijam o processamento automático por software de informações disponíveis na rede (Antoniou & Van Harmelen, 2004). Por exemplo, sistemas de gerenciamento de conhecimento, poderiam ter sua busca melhorada para ir além das pesquisas por palavras-chave, a qual apresenta diversos resultados para que o usuário filtre manualmente. Com os dados na Web Semântica, um programa poderia responder perguntas diretamente e apresentar uma única resposta correta para o usuário. A Web Semântica também pode ser usada na criação de assistentes pessoais que poderiam analisar diversas fontes de dados disponíveis na Web e tomar decisões de forma automática. Por exemplo, se você precisar ir a um médico, um assistente pessoal poderia analisar quais são os melhores médicos na especialidade que você precisa, que possuam consultório num raio de até cinco quilômetros de distância e que estejam disponíveis em até uma semana. Se necessário, o

assistente também poderia remarcar algum compromisso na sua agenda para marcar a consulta no médico. Tudo isso feito automaticamente por um software usando inteligência artificial e os dados disponíveis na Web Semântica.

A implementação da Web Semântica é dividida em camadas, cada uma com uma função bem definida (Koivunen & Miller, 2001). A organização das camadas é apresentada na Figura 2.2 e a seguir tem-se uma breve descrição dos objetivos de cada camada:

Unicode: representar e manipular textos em diferentes línguas (incluindo caracteres internacionais);

URI: prover meios para identificar os dados;

XML + NS + xmlschema: possibilitar a integração das definições da Web Semântica com padrões baseados em XML;

RDF + rdfschema: possibilitar a criação de definições sobre os dados, a criação de links que podem ser referenciados por URIs e a criação de tipos para os dados e os links;

Ontology vocabulary: definir o significado e as relações entre entidades;

Digital signature: identificar de forma segura quem escreveu um documento;

Logic: definir regras para permitir inferências;

Proof: executar as regras;

Trust: prover meios de identificação e de autorização.

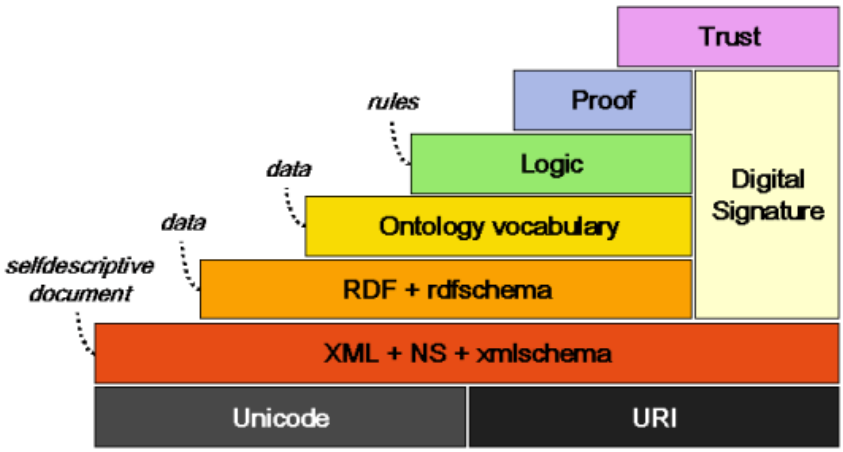

Figura 2.2: Camadas da Web Semântica

Fonte: (Koivunen & Miller, 2001)

Tabela 2.3: Correspondência entre construtores OWL e operadores DL

OWL	DL
EquivalentClass	$C \equiv D$
SubClassOf	$C \sqsubseteq D$
IntersectionOf	$C \sqcap D$
UnionOf	$C \sqcup D$
ComplementOf	$\neg C$
SomeValuesFrom	$\exists r.C$
AllValuesFrom	$\forall r.C$
Cardinality	$= n.R$
MinCardinality	$\geqslant n.R$
MaxCardinality	$\leqslant n.R$
Thing	$\top$
Nothing	$\bot$

2.1.3 Web Ontology Language

A *Web Ontology Language* (OWL) é uma linguagem de marcação com semântica formal baseada em lógica de descrição, utilizada para publicação e compartilhamento de ontologias na Web. Ela aparece na camada *ontology vocabulary* da web semântica e serve para descrever entidades e os seus relacionamentos. A OWL possibilita um engenheiro expressar conceitos mais complexos com semântica que vai além do que é possível apenas com o formato RDF usado para representar informações na Web (Dean *et al.* , 2004). Para isso, pode-se utilizar operadores lógicos da DL, como, por exemplo, interseção, união e negação, além de restrições existenciais, universais e de cardinalidade para relacionar conceitos e propriedades.

Atualmente a OWL está na sua versão 2 (OWL 2) (W3C, 2009), que é uma extensão da versão inicial, a OWL 1, que acrescenta novos recursos, mas mantém a compatibilidade entre as duas versões, ou seja, toda ontologia em OWL 1 é válida na versão 2. Entre os novos recursos tem-se o suporte a novos axiomas para aumentar a expressividade da linguagem, como união disjunta de classes, cadeias de propriedades, restrições de tipos de dados, etc.

A OWL possui sintaxe própria para representar os axiomas em DL. Usando construtores para representar os diferentes operadores da DL, para especificar classes, papéis e as relações entre eles. A Tabela 2.3 mostra a correspondência entre construtores da OWL e operadores da DL. Para exemplificar o uso da OWL, a Figura 2.3 mostra a codificação do axioma $Father \equiv Man \sqcap \exists hasChild.Person$.

Um perfil da OWL 2 é uma versão da OWL 2 mais simples que abre mão de expressividade para ganhar eficiência nas operações de raciocínio. Cada perfil tem sua utilidade dependendo do cenário em que é aplicado. Assim, um engenheiro deve escolher o perfil a ser utilizado de acordo com os requisitos da ontologia desenvolvida e como ela vai ser usada. A OWL 2 define três perfis: OWL 2 EL, OWL 2 QL e OWL 2 RL (Motik *et al.* , 2009), que são detalhados a seguir.

```
<EquivalentClasses>
    <Class IRI="#Father"/>
    <ObjectIntersectionOf>
        <Class IRI="#Man"/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="#hasChild"/>
            <Class IRI="#Person"/>
        </ObjectSomeValuesFrom>
    </ObjectIntersectionOf>
</EquivalentClasses>
```

Figura 2.3: Código OWL para representar o axioma $Father \equiv Man \sqcap \exists hasChild.Person$

OWL 2 EL é um subconjunto do OWL 2 no qual as operações de raciocínio possuem complexidade polinomial em relação ao tamanho da ontologia. Assim, esse perfil é útil para ontologias grandes com muitas classes e propriedades. Os algoritmos de raciocínio para esse perfil estão disponíveis em larga escala, possuindo implementações testadas e bem documentadas na literatura. A sigla EL está relacionada com a família $\mathcal{EL}$ da lógica de descrição.

OWL 2 QL é voltado para ontologias com um número muito grande de instâncias e que são utilizadas em aplicações que realizam muitas consultas aos seus dados. A expressividade desse perfil é pequena e operações de raciocínio como checagem de consistência e subsunção podem ser realizadas através de algoritmos com complexidade polinomial. A sigla QL está relacionada ao fato das consultas poderem ser realizadas como linguagens de consulta (*Query Language*) relacionais.

OWL 2 RL deve ser usado em ontologias que precisam de uma expressividade maior, mas que ainda necessitem do uso de operações de raciocínio rapidamente. Checagem de consistência, subsunções e checagem de instâncias, por exemplo, são realizadas em tempo polinomial. A sigla RL é usada pois o raciocínio pode ser implementado utilizando uma "*Rule Language*".

2.2 Questões de Competência

Questões de competência (QCs) (Gruninger & Fox, 1995) são um conjunto de perguntas que uma ontologia ou um conjunto de ontologias deve ser capaz de responder usando o conhecimento representado por seus axiomas. Assim, uma lista de QCs pode funcionar como a especificação de requisitos para ontologias, servindo para verificar se ela foi criada corretamente, ou seja, se contém todos os axiomas suficientes e necessários para responder corretamente as questões.

Muitos trabalhos propõem o uso de QCs em engenharia de ontologias. Neles, o processo de checagem de questões é frequentemente realizado de forma manual ou em alguns casos

fazendo a conversão das perguntas para consultas SPARQL (Zemmouchi-Ghomari & Ghomari, 2013). Entretanto, o processo de checagem manual é lento, custoso e pode ser impraticável para ontologias grandes ou quando a quantidade de QCs é extensa.

É importante esclarecer o conceito de QC utilizado nesse trabalho e quais suas semelhanças e diferenças com requisitos de software. Os requisitos de software vão desde descrições em alto nível (requisitos organizacionais e requisitos de produto (Aurum & Wohlin, 2005)) até testes executáveis, que também podem ser considerados requisitos, como no caso da abordagem de desenvolvimento orientado à testes (TDD) (Beck, 2003). Nesse trabalho, as QCs são perguntas que devem ser respondidas utilizando o conhecimento representado pelos axiomas da ontologia, ou que possa ser inferido a partir deles. Então, as QCs se assemelham mais com requisitos que são testes executáveis e que têm exigências claras para que eles sejam satisfeitos, do que com requisitos em mais alto nível.

Na Tabela 2.4 pode-se ver exemplos de QCs retiradas do trabalho de del Carmen Suárez-Figueroa *et al.* (2008) que foram usadas para uma ontologia que representa informações ligadas a serviços públicos de empregos. Já a Tabela 2.5 contém exemplos retirados do trabalho de Noy & McGuinness (2001) de perguntas que podem ser usadas para uma ontologia com informações sobre vinhos. Por fim, na Tabela 2.6, tem-se QCs apresentadas no trabalho de Staab *et al.* (2001) que foram usadas para uma ontologia sobre estratégia de negócios.

Tabela 2.4: Questões de competência para uma ontologia de serviços de empregos públicos

Questão de Competência	Tradução
What is the job seeker nationality?	Qual a nacionalidade do candidato ao emprego?
What is the job seeker desired job?	Qual o trabalho pretendido pelo candidato ao emprego?
What is the required work experience for the job offer?	Qual a experiência de trabalho necessária para a oferta de emprego?
When did the job seeker complete his/her first degree?	Quando o candidato ao emprego completou sua graduação?
What is the job seeker education level?	Qual o nível de escolaridade do candidato ao emprego?
Is the offered salary given in Euros?	O salário oferecido é dado em euros?

Fonte: del Carmen Suárez-Figueroa *et al.* (2008)

Tabela 2.5: Questões de competência para uma ontologia de vinhos

Questão de Competência	Tradução
Which wine characteristics should I consider when choosing a wine?	Quais características de um vinho eu devo considerar quando estiver escolhendo um vinho?
Is Bordeaux a red or white wine?	Bordeaux é um vinho tinto ou branco?
Does Cabernet Sauvignon go well with seafood?	Cabernet Sauvignon combina com frutos do mar?
What is the best choice of wine for grilled meat?	Qual a melhor escolha de vinho para acompanhar carne grelhada?
Which characteristics of a wine affect its appropriateness for a dish?	Quais características de um vinho afeta se ele é apropriado ou não para acompanhar um prato?
Does a bouquet or body of a specific wine change with vintage year?	O aroma ou corpo de um vinho específico muda de acordo com o ano que ele foi produzido?
What were good vintages for Napa Zinfandel?	Quais foram as boas safras para o Napa Zinfandel?

Fonte: Noy & McGuinness (2001)

Tabela 2.6: Questões de competência para uma ontologia de estratégia de negócios

Questão de Competência	Tradução
What are the subsidiaries, divisions and locations of company X?	Quais são as subsidiárias, divisões e localizações da empresa X
Which companies acquired company X?	Quais empresas adquiriram a empresa X?
Which companies merged in 1990 in the rubber industry?	Que empresas se fundiram em 1990 na indústria da borracha?
Who is CEO of company X?	Que é o CEO da empresa X?
Which activity of company X leads to operation in region Y?	Qual atividade de empresa X leva a operação na região Y?
Is there any regional expansion of company X due to the acquisition of company Y?	Existe alguma expansão regional da empresa X, devido à aquisição da empresa Y?

Fonte: Staab *et al.* (2001)

2.3 Engenharia de Ontologias

De acordo com Gómez-Pérez *et al.* (2004), engenharia de ontologias se refere às atividades relacionadas ao processo, ciclo de vida, métodos, metodologias, ferramentas e linguagens para suportar o desenvolvimento de ontologias. Devedzić (2002) definiu que engenharia de ontologias deve cobrir o conjunto de atividades realizadas durante a conceitualização, projeto, implementação e implantação de uma ontologia. Já Mizoguchi (1998) definiu que o propósito da engenharia de ontologias deve cobrir assuntos relacionados a filosofia, representação de conhecimento, projeto de ontologias, padronização, reuso e compartilhamento de conhecimento.

De certa forma, as metodologias para desenvolvimento de ontologias se assemelham com às metodologias utilizadas para construir software. Ambas fornecem orientações para os engenheiros que usualmente são divididas em fases, por exemplo, especificação, execução e avaliação. Além disso, os processos de construção de ontologias costumam seguir uma abordagem iterativa e incremental, que é uma abordagem consolidada na engenharia de software. Ontologias e software podem evoluir de forma parecida durante a sua vida, por isso, ambos podem requerer atividades relacionadas a manutenção, versionamento, rastreabilidade, entre outros.

O Capítulo 3 traz mais informações sobre metodologias de desenvolvimento de ontologias. Nele são apresentados detalhes de quatro metodologias diferentes, deixando claras as atividades que devem ser desenvolvidas e os artefatos que são produzidos.

2.4 Rastreabilidade de Requisitos

Nesta seção são abordados conceitos sobre rastreabilidade de requisitos em duas áreas distintas: engenharia de software e engenharia de ontologias.

2.4.1 Rastreabilidade de Requisitos em Engenharia de Software

Gotel & Finkelstein (1994) definiram rastreabilidade de requisitos como a habilidade de descrever e seguir o ciclo de vida de um requisito, tanto no sentido que vai da origem do requisito, passando por sua especificação, desenvolvimento, até a sua implantação e uso, quanto no sentido contrário. Outra definição para rastreabilidade de requisitos foi dada por Pinheiro (2004): a habilidade de definir, capturar, e seguir os rastros deixados pelos requisitos em outros artefatos do desenvolvimento de software e os rastros deixados por esses artefatos nos requisitos.

O suporte a rastreabilidade de requisitos é importante sempre que sistemas complexos precisam ser mantidos, podendo ser crítico para o sucesso do projeto (Asuncion *et al.* , 2007). A rastreabilidade também é essencial para o reuso e evolução de software. As atividades relacionadas à rastreabilidade de requisitos ajudam a manter o software, a evitar inconsistência e omissões relacionadas a implementação dos requisitos, a mostrar a evolução dos artefatos e a

dar certeza para as pessoas envolvidas no projeto que todos os requisitos estão sendo satisfeitos corretamente (Winkler & Pilgrim, 2010).

Por outro lado, a rastreabilidade de requisitos tem um custo alto de manutenção e, assim como a documentação de um projeto, muitas vezes os rastros deixam de ser atualizados e acabam sendo esquecidos, principalmente quando essa atividade é feita manualmente. A automação da captura e manutenção dos rastros contribui para diminuição de custos e evita os problemas de esquecimento de atualização (Cleland-Huang *et al.* , 2005), entretanto isto é possível apenas parcialmente ou em contextos restritos.

Diferentes tipos de artefatos podem ser relacionados aos requisitos para que eles sejam rastreáveis. Um mudança na especificação de requisitos pode impactar em várias partes do projeto, por exemplo, na documentação, nos testes e no código. O oposto também é verdade, ou seja, mudanças nos artefatos também podem impactar na especificação de requisitos. Assim, o rastreamento de requisitos pode ser usado para informar aos integrantes de um projeto o impacto das mudanças em diferentes tipos de artefatos. Em sistemas grandes e complexos isto é ainda mais importante, pois se torna difícil alguém conhecer em detalhes o sistema inteiro e conseguir prever com exatidão todos os artefatos que precisam ser alterados ao realizar alguma mudança no projeto.

Muitas ferramentas nessa área funcionam de forma semelhante. Basicamente, elas armazenam os artefatos, que podem ser anotados com metadados, e criam relacionamentos entre eles. Além disso, essas ferramentas costumam fornecer meios para que os usuários consigam visualizar as informações do rastreamento, por exemplo, matrizes de rastreabilidade, diagramas com referências cruzadas ou grafos (Wieringa, 1995). Uma das ferramentas mais populares para gerenciar requisitos atualmente é o IBM Rational DOORS (Saiedian *et al.* , 2013).

A Tabela 2.7 traz um exemplo de uma matriz de rastreabilidade (Wiegers & Beatty, 2013). Nela é possível observar que os requisitos funcionais estão ligados a casos de uso, elementos de projeto, código e testes. Um elemento de projeto pode ser uma classe ou uma tabela em um banco de dados, por exemplo. Já o elemento de código está ligado ao código fonte que vai ser implementado, seja ele um método, uma função, etc.

Tabela 2.7: Exemplo de matriz de rastreabilidade

Caso de uso	Requisito funcional	Elemento de projeto	Elemento de código	Teste
UC - 28	catalog.query.sort	Class catalog	CatalogSort()	search.7 search.8
UC - 29	catalog.query.import	Class catalog	CatalogImport() CatalogValidate()	search.12 search.13 search.14

Apesar das várias soluções desenvolvidas, muitas ainda precisam de intervenção manual dos engenheiros de software para funcionar. Isto resulta em um grande esforço e no aumento do complexidade do projeto (Ramesh *et al.* , 1995). Por isso, várias pesquisas têm se concentrado em

criar ferramentas para ajudar e automatizar as atividades envolvidas no rastreamento de requisitos. Diversas técnicas são empregadas nesses trabalhos. Alguns usam abordagens baseada em regras. Por exemplo, Egyed & Grünbacher (2005) usa informações da execução de testes e alguns relacionamentos entre artefatos criados manualmente para inferir novos relacionamentos, e Gervasi & Zowghi (2005) propuseram um método para identificar inconsistências transformando os requisitos expressados em linguagem natural em lógica formal. Outra abordagem comum é o uso de técnicas de recuperação de informação. Por exemplo, RETRO (Hayes *et al.* , 2007), Poirot (Lin *et al.* , 2006) e ADAMS (De Lucia *et al.* , 2008) são ferramentas que processam o texto de documentos e usam modelos matemáticos para sugerir novos relacionamentos entre os artefatos.

É importante observar que nas ferramentas apresentadas anteriormente, não é possível completamente efetuar todas as atividades necessárias para dar suporte a rastreabilidade de requisitos sem a intervenção humana.

2.4.2 Rastreabilidade de Requisitos em Engenharia de Ontologias

Rastreamento de requisitos em engenharia de ontologias é a habilidade de descrever e de seguir o ciclo de vida das questões de competência (requisitos da ontologia), tanto no sentido que vai da origem da QC, passando por sua especificação, desenvolvimento, implantação e manutenção, quanto no sentido contrário.

A definição é similar à definição de rastreamento de requisitos na engenharia de software. Isto se deve ao fato de que o propósito de ambos é o mesmo. Por outro lado, os artefatos rastreados são diferentes, assim como as formas que os relacionamentos entre eles podem ser descobertos e armazenados. Na engenharia de ontologias, os requisitos são as QCs e o número de artefatos que podem ser relacionados ainda é pequeno. Talvez no futuro, com o progresso da engenharia de ontologias, os processos fiquem mais complexos e mais artefatos sejam produzidos. Além disso, na engenharia de ontologias, pode-se trabalhar com ontologias em lógica de descrição. Esse tipo de artefato possui uma base de lógica formal que fornece possibilidades de processamento automático, incluindo o processamento voltado para o rastreamento de requisitos, que é difícil de replicar na engenharia de software.

Assim como na engenharia de software, a rastreabilidade de requisitos na engenharia de ontologias tem o potencial para melhorar a qualidade dos artefatos produzidos. Especialmente em ontologias muito grandes, que podem ser tão complexas quanto grandes sistemas de software. A rastreabilidade de requisitos em engenharia de ontologias pode ajudar os engenheiros a se certificarem que todos os requisitos são satisfeitos, mostrar o impacto de uma mudança em um artefato nos outros artefatos do projeto, melhorar as tarefas de manutenção e monitorar de forma efetiva o progresso na construção de ontologias. Até aqui pode-se perceber uma semelhança grande entre os dois tipos de rastreabilidade de requisitos, entretanto, para ontologias, as atividades de rastreamento podem ser mais eficientes.

Trabalhos recentes apresentaram formas de checar questões de competência escritas em linguagem natural, que podem ser usadas como requisitos, automaticamente (Bezerra *et al.* , 2013) (Malheiros & Freitas, 2014). Assim, é possível usar os requisitos para checar automaticamente se uma ontologia satisfaz tudo o que deveria. Isto é muito difícil de ser replicado na engenharia de software, pois existe um nível de abstração maior entre os requisitos e o código implementado. Também, a relação oposta pode ser encontrada, ou seja, a partir de um axioma, visualizar as QCs relacionadas a ele, basta que as perguntas em linguagem natural sejam interpretadas como axiomas em DL. Ambiguidade pode acontecer, pois é inerente da linguagem natural, mas seguindo convenções ou seguindo um processo de desenvolvimento bem definido, é possível automatizar a criação de relações entre QCs e os axiomas da ontologia.

De acordo com as classificações apresentadas por Gotel & Finkelstein (1994), o trabalho apresentado nesta tese é focado no rastreamento pós-especificação de requisitos. Em outras palavras, ele é focado nas fases que estão relacionadas à implementação dos requisitos, ou, no caso da engenharia de ontologias, fases relacionadas à adição de axiomas que satisfaçam as QCs. O rastreamento pré-especificação de requisitos não foi levado em consideração, mas deve ser investigado no futuro.

Apesar de pouco explorada na literatura, algumas metodologias, como a On-To-Knowledge (OTKM), apresenta características seminais de rastreabilidade de requisitos na engenharia de ontologias. Um dos documentos que deve ser produzido pelos engenheiros usando a metodologia OTKM é um questionário com as questões de competências. Nele, além da lista das questões, os engenheiros devem especificar os nomes dos conceitos e relações envolvidas em cada QC, assim, criando um mecanismo manual de rastreabilidade. Mesmo simples e limitada, a rastreabilidade de requisitos na OTKM traz indícios da importância de tais atividades no processo de desenvolvimento de ontologias.

2.5 Conclusão

Este capítulo explicou diversos conceitos que são utilizados no restante do trabalho e que precisam ser compreendidos antes de serem apresentadas as principais contribuições realizadas.

Foram dados conceitos e exemplos relacionados à ontologias, lógica de descrição, web semântica e OWL . Em seguida foram introduzidas as questões de competência e a engenharia de ontologias. Por fim, foi mostrado o que é rastreabilidade de requisitos, tanto na visão da engenharia de software, quanto na visão da engenharia de ontologias.

O próximo capítulo aborda diversas metodologias para engenharia de ontologias, detalhando suas etapas e artefatos gerados.

3

Metodologias para Engenharia de Ontologias

Neste capítulo são apresentadas quatro metodologias para engenharia de ontologias, que visam organizar o desenvolvimento de ontologias propondo etapas, práticas, documentos a serem produzidos e ciclos de vida.

3.1 Uschold & Gruninger

No trabalho apresentado por Uschold & Gruninger (1996) foi proposta uma metodologia para desenvolvimento de ontologias contendo as seguintes etapas:

- Identificação do propósito e do escopo;
- Construção da ontologia;
- Avaliação;
- Documentação.

A seguir cada etapa é apresentada em detalhes.

3.1.1 Identificar o Propósito e o Escopo

O primeiro passo da metodologia é definir por qual motivo a ontologia está sendo criada e quais são os casos de uso esperados para ela. Também é importante identificar os usuários potenciais da ontologia.

Por exemplo, propósitos comuns para ontologias encontradas na literatura são: servir como base de conhecimento estruturado sobre um determinado domínio e definir um vocabulário comum utilizado para um conjunto de aplicações.

3.1.2 Construir a Ontologia

Após identificar o propósito e o escopo, tem-se um objetivo bem definido para dar início à construção da ontologia. Nessa etapa existem três aspectos principais: captura, codificação e integração de ontologias existentes.

A captura consiste na identificação dos conceitos chaves que serão representados na ontologia e suas relações, na produção de descrições não ambíguas desses conceitos e relações, e, por fim, na indicação de termos que poderão ser usados para identificar cada conceito ou relação.

A codificação é a representação explícita do que foi capturado anteriormente. Isso envolve se comprometer com os nomes dos termos definidos, escolher uma linguagem para representar a ontologia e escrever o código. Muitas vezes a codificação e captura ocorrem ao mesmo tempo, ou seja, os engenheiros que estão criando a ontologia podem identificar conceitos e suas relações e já escrever essas definições usando uma linguagem de representação de ontologias.

Durante todo o processo de construção sempre deve ser pensado na possibilidade do reuso de ontologias existentes. Os autores apontam que essa não é uma atividade fácil, e mostram que progressos nessa área podem ser vistos nos trabalhos de (Farquhar *et al.* , 1995) e (Skuce, 1995).

3.1.3 Avaliação

A ontologia deve ser avaliada de acordo com alguma referência para que seja possível fazer um julgamento técnico se a ontologia está correta ou não. Essa referência pode ser a especificação de requisitos, questões de competências ou simplesmente o uso da ontologia no mundo real.

A etapa de avaliação está alinhada com o método de expansão proposto nessa tese. As questões de competência checadas automaticamente pelo sistema servem de referência para saber se a ontologia está correta ou não.

3.1.4 Documentação

Na metodologia é desejável que existam diretrizes para documentação da ontologia que estiver sendo construída. Elas não precisam ser sempre iguais, as diretrizes podem mudar de acordo com o tipo e propósito da ontologia construída.

A falta de documentação adequada é uma das barreiras apontadas por Skuce (1995) para o compartilhamento efetivo de conhecimento. Por isso, é importante documentar as suposições tomadas em relação aos conceitos definidos e suas relações.

3.2 Methontology

Methontology é uma metodologia proposta por Fernandez-Lopez *et al.* (1997) para construção de ontologias desde o seu início. Ela foi baseada na experiência adquirida pelos seus criadores durante o desenvolvimento de uma ontologia de produtos químicos. Até a data de publicação do trabalho sobre a metodologia, muitas ontologias já tinham sido propostas, entretanto poucas publicações traziam instruções de como construir esse tipo de artefato. Dessa forma, o processo de construção de uma ontologia era muito mais artesanal do que uma atividade da engenharia. Para solucionar esse problema era preciso desenvolver atividades padronizadas, definir o ciclo de vida de uma ontologia e criar ferramentas para apoiar o processo.

A Methontology define um conjunto de fases que devem fazer parte do processo de construção de uma ontologia, assim como as técnicas mais apropriadas para cada fase e os artefatos produzidos. As fases da metodologia são:

- Especificação;
- Aquisição de conhecimento;
- Conceitualização;
- Integração;
- Implementação;
- Avaliação;
- Documentação.

A seguir são apresentadas em detalhes cada uma das fases da metodologia Methontology.

3.2.1 Especificação

O objetivo da fase de especificação é produzir um documento de especificação da ontologia. As informações mínimas que devem estar contidas no documento são:

- O propósito da ontologia, cenários de uso e quem são os seus usuários finais;
- Nível de formalidade usado para implementação da ontologia: muito informal, semi-informal, semi-formal ou rigorosamente formal;
- Escopo, que inclui o conjunto de termos representados, suas características e granularidade.

A Figura 3.1 apresenta um exemplo de um documento de especificação.

Documento de especificação da ontologia	
Domínio	Produtos químicos
Data	15 de maio de 2014
Conceitualização	João da Silva
Implementação	José da Silva
Propósito	Ontologia sobre substâncias químicas que será usada quando informações sobre elementos químicos forem necessárias em atividades de ensino, manufatura, análise, etc. Essa ontologia pode ser usada para obter, por exemplo, a massa atômica de elemento Sódio.
Nível de formalidade	Semi-formal
Escopo	Lista de 103 elementos: Lítio, Sódio, … Lista de conceitos: Halogênio, gás nobre, semi-metal, metal, … No mínimo informações sobre as seguintes propriedades: número atômico, massa atômica, volume atômico a 20 graus celsius, ponto de ebulição, densidade a 20 graus celsius, eletronegatividade, afinidade eletrônica e símbolo
Fonte de conhecimento	Handbook of Chemistry and Physics. 65th edition. CRC-Press, Inc. 1984-1985.

Figura 3.1: Exemplo de documento de especificação

3.2.2 Aquisição de Conhecimento

A fase de aquisição de conhecimento acontece em paralelo às outras fases do processo. Sendo a maior parte da aquisição realizada simultaneamente à especificação de requisitos e o restante tendendo a se espalhar de forma decrescente durante as próximas fases do processo. Esta fase é análoga a elicitação de requisitos em engenharia de software, que tem como objetivo obter conhecimento sobre um domínio para entender e especificar o que um software necessita.

As seguintes técnicas foram usadas pelos autores da Methontology para adquirir conhecimento:

- Entrevistas não estruturadas com especialistas para criar um rascunho do documento de requisitos;
- Análise informal de livros e manuais para conhecer os principais conceitos da área;
- Análise formal de textos para aprofundar o conhecimento de conceitos e para descobrir relações entre os conceitos;
- Entrevistas estruturadas com especialistas para conseguir conhecimento detalhado sobre os conceitos, suas propriedades e relações, e para avaliar o modelo conceitual após a fase de conceitualização.

3.2.3 Conceitualização

Nessa fase, o conhecimento obtido é estruturado num modelo conceitual. A primeira etapa a ser realizada é a construção de um glossário de termos, estes termos incluem conceitos, propriedades, instâncias e verbos. Assim, o glossário de termos reúne todo o conhecimento obtido que potencialmente será usado na ontologia e os seus significados. Após a construção do glossário, os engenheiros devem separar os conceitos e verbos. Os autores recomendam diretrizes específicas para trabalhar com o conjunto de conceitos e com o conjunto de verbos.

3.2.4 Integração

Com o objetivo de agilizar a construção de uma ontologia, os engenheiros devem considerar o reuso de definições já especificadas em outras ontologias. Para isso, primeiro devem ser analisadas meta-ontologias (Cyc (Lenat, 1995), Ontolingua (Gruber, 1992), etc.) e em seguida bibliotecas de ontologias disponíveis. Meta-ontologias são ontologias mais genéricas, com conceitos e relações que auxiliam a construção, projeto e organização de outras ontologias.

Um documento de integração é gerado como resultado dessa fase. Ele deve conter as seguintes informações:

- Qual meta-ontologia está sendo usada;
- Uma lista com todos os termos reusados, a qual ontologia ele pertence e qual o seu nome no modelo conceitual.

A Figura 3.2 mostra um exemplo de um documento de integração.

Documento de integração		
Meta-ontologia	Ontolingua	
Nome do termo na conceitualização	**Ontologia reusada**	**Nome do termo na ontologia**
Kilometer	Ontolingua Standard-Units	Kilometer
Centimeter	Ontolingua Standard-Units	Centimeter
Exponent	Ontolingua KF-Numbers	Expt

Figura 3.2: Exemplo de documento de integração

3.2.5 Implementação

Nesta fase, os autores argumentam que a implementação da ontologia requer um ambiente de desenvolvimento que suporte o reuso de ontologias e tenha uma série de funcionalidades como: analisador sintático, editor, navegador e sistema de busca.

O resultado dessa fase é uma ontologia codificada em uma linguagem formal. Os autores citam CLASSIC, BACK, LOOM, Ontolingua, Prolog e C++, mas podemos estender a ideia para linguagens mais modernas como RDF e OWL.

3.2.6 Avaliação

Na fase de avaliação os engenheiros fazem um julgamento técnico para saber se a ontologia desenvolvida está satisfazendo os requisitos levantados. O resultado dessa fase é um conjunto de documentos de avaliação, nos quais os engenheiros descrevem como a ontologia foi avaliada, as técnicas usadas, os tipos de erros encontrados e as fontes de conhecimento utilizadas na avaliação.

3.2.7 Documentação

A fase de documentação é mais uma fase que deve ser realizada em paralelo a outras fases, assim como a fase de aquisição de conhecimento. Ao avaliar os artefatos gerados nas fases descritas anteriormente, pode-se perceber que durante o ciclo de vida da ontologia vários documentos são gerados. A Methontology incorpora a criação de documentação como prática natural do desenvolvimento, assim solucionando um problema apontado pelos autores da falta de orientações para se criar documentação durante o desenvolvimento de ontologias.

3.3 On-To-Knowledge

O projeto On-To-Knowledge (Sure *et al.* , 2004) construiu um conjunto de ferramentas para auxiliar o desenvolvimento de ontologias. Nele foi desenvolvida uma linguagem para especificação de ontologias chamada OIL e também foi definida uma metodologia de construção de ontologias. Assim, os pesquisadores do projeto reuniram seus esforços para melhorar e organizar diversas áreas relacionadas a engenharia de ontologias.

A metodologia proposta pelo projeto On-To-Knowledge possui as etapas:

- Estudo de viabilidade;
- Pontapé inicial;
- Refinamento;
- Avaliação;
- Aplicação e evolução.

A seguir cada etapa é apresentada com mais detalhes.

3.3.1 Estudo de Viabilidade

A primeira etapa da metodologia é a realização de um estudo de viabilidade. Seu objetivo é analisar fatores que vão além de questões tecnológicas, pois eles também podem ser fundamentais para o sucesso do uso de uma ontologia em um sistema. Esse estudo serve para ajudar na tomada de decisões relacionadas a viabilidade econômica e técnica do projeto e para identificar problemas, oportunidades e possíveis soluções.

3.3.2 Pontapé Inicial

Na segunda fase da metodologia é produzido o documento de especificação de requisitos. Ele contém informações iniciais da área de aplicação da ontologia e lista fontes de conhecimento que serão usadas no desenvolvimento. A principal função do documento de especificação de requisitos é guiar os engenheiros a decidirem sobre a inclusão e exclusão de conceitos e relações, assim como auxiliar na definição da hierarquia da ontologia.

Além do documento de especificação de requisitos, nessa fase, também é produzida uma especificação semi-formal da ontologia, isto é, um grafo com os conceitos importantes e as suas relações. Cada conceito ou relação podem estar acompanhados de uma descrição textual para fornecer mais informações para os engenheiros.

3.3.3 Refinamento

Nessa fase, a especificação semi-formal criada na fase anterior é refinada para que seja construída uma ontologia usando uma linguagem de especificação de ontologias, no caso do projeto On-To-Knowledge, a linguagem OIL. O refinamento pode ser feito de forma *top-down*, *middle-out* ou *bottom-up*. Na abordagem *top-down* começa-se com conceitos e relações mais genéricas que em seguida são refinados e detalhados. Usando uma estratégia *middle-out*, inicialmente identifica-se os conceitos e relações mais importantes para que depois os engenheiros completem a hierarquia através de generalização e especialização desses conceitos e relações. Por fim, usando um refinamento *bottom-up* os engenheiros partem de conceitos mais específicos e vão generalizando ao longo do tempo. Os autores do projeto On-To-Knowledge argumentam que a abordagem *bottom-up* pode ser aplicada com ferramentas de análise automática de documentos.

Essa etapa é finalizada quando os engenheiros decidem que a ontologia construída satisfaz todos os requisitos definidos na fase de pontapé inicial. Essa decisão é tomada através da experiência dos engenheiros envolvidos, pois eles precisam comparar manualmente a ontologia criada com o documento de especificação de requisitos.

3.3.4 Avaliação

Existem três tipos diferentes de avaliações na metodologia: avaliação com foco na tecnologia, avaliação com foco no usuário e avaliação com foco na ontologia. A primeira, a

avaliação com foco na tecnologia, tem como objetivo avaliar se a especificação da ontologia está seguindo todas as regras de sintaxe da linguagem escolhida para sua construção, se a ontologia é consistente, se ela é interoperável e se ela é escalável. Já a avaliação com foco no usuário tem como objetivo avaliar se os usuários da aplicação que usa a ontologia estão satisfeitos. Por fim, a avaliação com foco na ontologia está preocupada em avaliar a estrutura da ontologia, ou seja, com sua hierarquia e o significado pretendido para as classes e papeis, e as relações entre eles. Os autores do projeto On-To-Knowledge citam, para essa avaliação, o uso da abordagem proposta por Guarino & Welty (2002).

Ao final dessa etapa é produzida uma ontologia avaliada nos três focos apresentados anteriormente. Entretanto, é esperado que um ciclo avaliação-refinamento-avaliação seja repetido até que seja decidido que a ontologia satisfaz todos os critérios de avaliação.

3.3.5 Aplicação e Evolução

Nessa fase, a ontologia se encontra em uso em uma aplicação em produção. Como muitos sistemas de software, a especificação inicial pode necessitar de alterações, com isso, a ontologia deve refletir as mudanças exigidas. Entretanto, uma nova versão da ontologia só deve ser liberada após testes cuidadosos para avaliar os efeitos das mudanças na aplicação.

3.4 NeOn

Diferente de outros trabalhos que propõem metodologias para construção de ontologias, a metodologia NeOn (Suarez-Figueroa *et al.* , 2012) não descreve um processo fixo, ao invés disso, são propostos diferentes caminhos que podem ser seguidos no processo de desenvolvimento de acordo com a natureza do projeto. A NeOn é uma metodologia baseada em cenários, que suporta reuso e evolução ao longo do tempo.

A metodologia apresenta quatro principais contribuições:

- Um glossário de processos e atividades envolvidos no desenvolvimento de ontologias;
- Descrição de nove cenários possíveis na construção de ontologias;
- Dois modelos de ciclo de vida para ontologias;
- Guias metodológicos para diferentes processos e atividades.

Nas próximas subseções cada contribuição é apresentada em detalhes.

3.4.1 Glossário

O Glossário da metodologia NeOn identifica e define processos e atividades que costumam estar envolvidos no desenvolvimento de ontologias (Suárez-Figueroa, 2010). Ele inclui 59 processos e atividades que são listados nas Tabela 3.1 e 3.2.

Tabela 3.1: Processos do glossário da metodologia NeOn

Processos	
Alinhamento de ontologias	Reuso de recursos não ontológicos
Reuso de padrão de projeto de ontologias	Reuso de recursos ontológicos
Reuso de módulo de ontologia	Reuso de ontologia
Reengenharia de ontologia	Reuso de definições de ontologia

Tabela 3.2: Atividades do glossário da metodologia NeOn

Atividades	
Anotação de ontologias	*Merge* de ontologias
Avaliação (focada no usuário) de ontologias	Modificação de ontologias
Comparação de ontologias	Modularização de ontologias
Conceitualização de ontologias	Extração de módulos de ontologias
Controle de configuração de ontologias	Particionamento de ontologias
Customização de ontologias	Povoamento de ontologias
Diagnóstico de ontologias	Poda de ontologias
Documentação de ontologias	Controle de qualidade de ontologias
Elicitação de ontologias	Reparo de ontologias
Enriquecimento de ontologias	Especificação de requisitos de ontologias
Estudo do ambiente de ontologias	Engenharia reversa de recursos não ontológicos
Avaliação técnica de ontologias	Transformação de recursos não ontológicos
Evolução de ontologias	Reestruturação de ontologias
Extensão de ontologias	Engenharia reversa de ontologias
Estudo de viabilidade de ontologias	Agendamento
Formalização de ontologias	Busca de ontologias
Engenharia *forward* de ontologias	Seleção de ontologias
Implementação de ontologias	Especialização de ontologias
Integração de ontologias	Sumarização de ontologias
Aquisição de conhecimento para ontologias	Tradução de ontologias
Aprendizagem de ontologias	Melhoria de ontologias
Localização de ontologias	Atualização de ontologias
Mapeamento de ontologias	Verificação de ontologias
Matching de ontologias	Versionamento de ontologias

Os trabalhos (Suárez-Figueroa, 2010) e (Suarez-Figueroa *et al.* , 2012) descrevem com mais detalhes cada uma das 59 atividades ou processos do glossário.

3.4.2 Nove Cenários para Construção de Ontologias

A NeOn identifica nove possíveis cenários para o desenvolvimento de ontologias. São eles:

Da especificação à implementação: a ontologia é construída do zero, ou seja, sem reuso de nenhum recurso;

Reuso e reengenharia de recursos não ontológicos: os engenheiros precisam analisar recursos não ontológicos para decidir quais desses recursos podem ser reusados e para realizar a reengenharia deles, transformando o conhecimento obtido em ontologias;

Reuso de recursos ontológicos: os engenheiros reusam ontologias completas, módulos ou definições de uma ontologia no desenvolvimento de uma nova ontologia;

Reuso e reengenharia de recursos ontológicos: nesse cenário, os engenheiros além de reusar, também fazem reengenharia de recursos ontológicos;

Reuso e *merging* de recursos ontológicos: esse cenário aparece quando muitos recursos ontológicos de um mesmo domínio são escolhidos para reuso, assim os engenheiros podem criar uma nova ontologia a partir dos recursos escolhidos para reuso;

Reuso, *merging* e reengenharia de recursos ontológicos: esse cenário é similar ao cenário anterior, mas os engenheiros realizam reengenharia da ontologia criada através do *merge* dos recursos ontológicos reusados;

Reuso de padrões de projeto de ontologias: os engenheiros acessam repositórios de padrões de projetos de ontologias (Gangemi & Presutti, 2009) para reusá-los;

Reestruturação de recursos ontológicos: os engenheiros reestruturam recursos ontológicos que são integrados na ontologia em desenvolvimento, ou seja, eles criam módulos, podam, estendem e especializam de acordo com as definições contidas nos recursos usados;

Localização de recursos ontológicos: nesse cenário, os engenheiros adaptam uma ontologia para que ela seja utilizada por pessoas que entendem outros idiomas.

Os cenários apresentados podem ser combinados de diversas maneiras de acordo com a necessidade da equipe de desenvolvimento de ontologias, apenas com a exigência de que o primeiro cenário (da especificação à implementação) sempre deve estar presente. Além disso, as atividades de aquisição de conhecimento, documentação, gerência de configuração e avaliação devem ser realizadas durante todo o processo de desenvolvimento de ontologias em qualquer um dos cenários apresentados.

3.4.3 Dois Modelos de Ciclo de Vida para Ontologias

O processo de desenvolvimento de ontologias é similar ao processo de desenvolvimento de software. Em ambos, artefatos são desenvolvidos para satisfazer um conjunto de requisitos. Na NeOn, o ciclo de vida de uma ontologia é definido através da organização dos processos e atividades contidas no glossário.

Foram definidos dois modelos de ciclo de vida na metodologia: o modelo cascata e o modelo iterativo-incremental.

3.4.3.1 Modelo Cascata

O modelo cascata divide o ciclo de vida da ontologia em etapas que devem ser realizadas sequencialmente. Ou seja, uma etapa deve ser completada totalmente antes dos engenheiros seguirem para a etapa seguinte. Também não é permitido voltar etapas, exceto para a etapa de manutenção. Esse modelo deve ser usado quando todos os requisitos são conhecidos e imutáveis desde o começo do desenvolvimento.

A NeOn permite que o modelo cascata tenha algumas variações de acordo com as atividades de reuso e reengenharia que precisarem ser realizadas, no total o trabalho apresenta cinco variações do modelo cascata. A Figura 3.3 mostra o modelo mais simples, com quatro fases sequenciais e com a possibilidade de retorno da fase de manutenção para a fase de projeto.

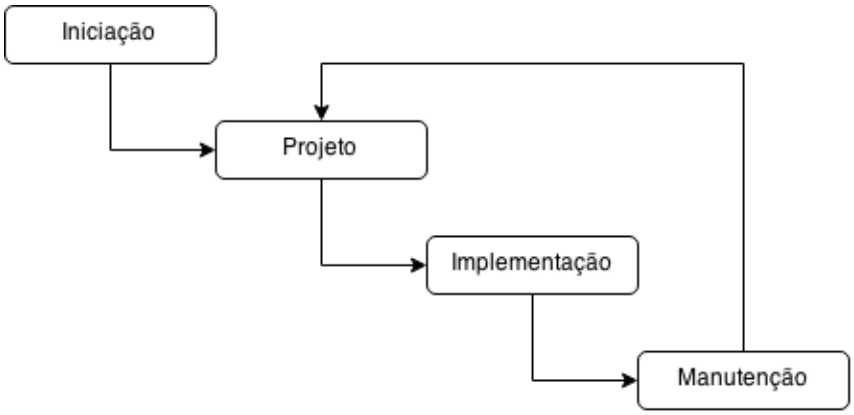

Figura 3.3: Modelo cascata de ciclo de vida de ontologias

3.4.3.2 Modelo Iterativo-incremental

O modelo iterativo-incremental organiza o desenvolvimento em iterações com duração fixa. Cada iteração tem a estrutura parecida com o modelo cascata, como pode ser visto na Figura 3.4. Esse modelo deve ser usado em projetos com equipes grandes, envolvendo diferentes domínios e quando os requisitos não são completamente definidos no início do projeto ou podem mudar ao longo do período de desenvolvimento.

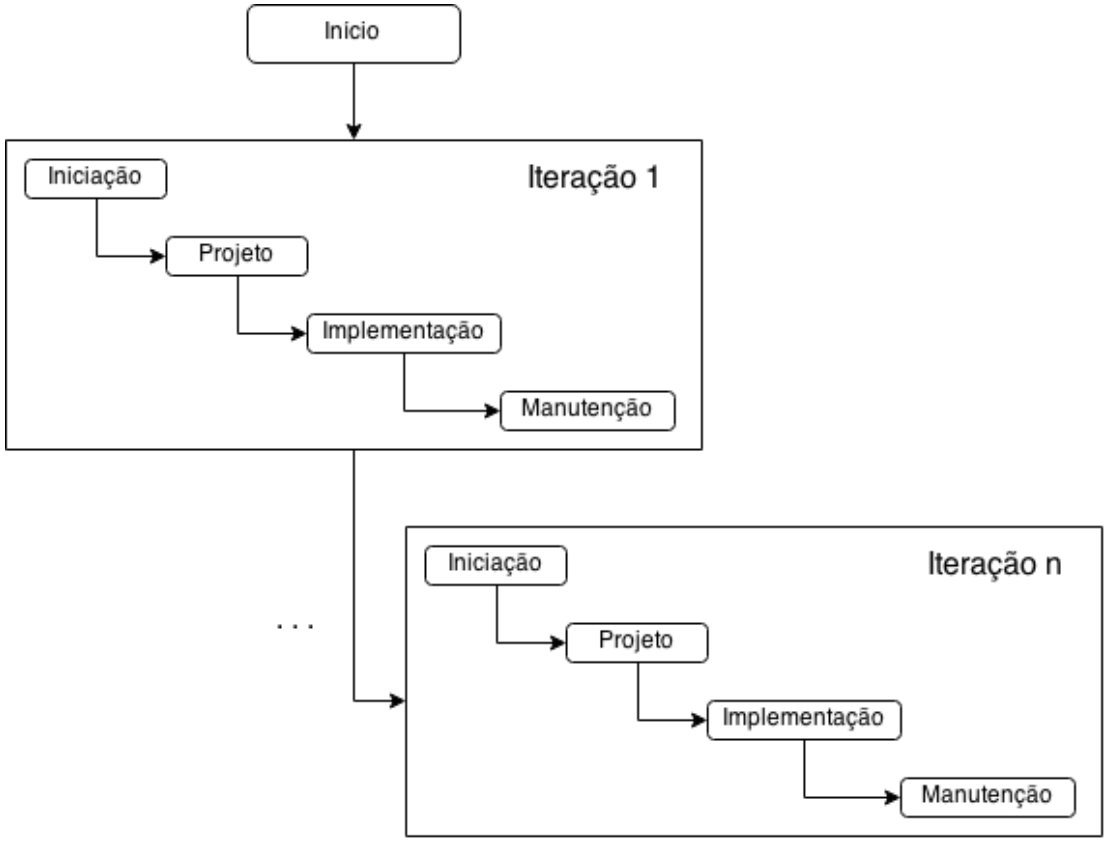

Figura 3.4: Modelo iterativo-incremental de ciclo de vida de ontologias

3.4.4 Guias Metodológicos

A metodologia NeOn também apresenta guias metodológicos para alguns dos processos e atividades do seu glossário. Para cada um é apresentada sua definição, o objetivo, a entrada, ou seja, os recursos necessários para realizar o processo ou atividade, a saída (os resultados), as pessoas envolvidas e quando o processo ou atividade deve ser realizado. Também é provido para os engenheiros um gráfico que mostra o fluxo do processo ou atividade. Por fim, exemplos são fornecidos para que os engenheiros tenham amostras concretas de como realizar o processo ou atividade corretamente. A metodologia apresenta guias para cada um dos cenários descritos anteriormente, além de guias para avaliação e evolução das ontologias.

3.5 Metodologias Ágeis

Assim como na engenharia de software, metodologias ágeis surgiram no escopo da engenharia de ontologias. Essas metodologias tem como principais características a flexibilidade e a capacidade de reagir eficientemente a mudanças de requisitos.

A metodologia XP.K (Knublauch, 2002) foi uma das primeiras a abordar os princípios do desenvolvimento ágil na área de ontologias. O XP.K aplica os valores da Extreme Programming (Beck, 2000) no desenvolvimento. São eles: comunicação, simplicidade, *feedback* e coragem.

Entretanto, ao invés de comunicação, a metodologia usa um princípio mais abrangente chamado comunidade. Durante o desenvolvimento, os engenheiros de ontologias colaboram diretamente com os especialistas no domínio que está sendo modelado, ambos guiados pelo valor da humildade. Tal colaboração é fundamental, pois os especialistas possuem o conhecimento necessário para especificar e avaliar a ontologia e os engenheiros possuem o conhecimento para criar o artefato. As ontologias no XP.K são desenvolvidas utilizando orientação a objetos, que provê meios para representar classes, atributos, relações e restrições.

A RapidOWL (Auer & Herre, 2006) é uma metodologia ágil para engenharia de conhecimento colaborativa, que envolve o desenvolvimento de ontologias. Ela é inspirada pela metodologia XP.K e pelos Wikis (Leuf & Cunningham, 2001), que trazem uma forma de edição de texto colaborativa de sucesso comprovado. A RapidOWL também adota os valores da Extreme Programming, mas ela combina os valores comunicação e *feedback* em um só chamado comunidade, além de adicionar o valor transparência, que especifica que todas as atividades devem ser observadas e revisadas por todos os envolvidos no processo. Na metodologia, o desenvolvimento de uma ontologia é guiado pelos princípios: suposição de mundo aberto, incentivo a mudanças incrementais, métodos de criação uniforme, desenvolvimento observável e *feedback* rápido. Com relação as práticas, a RapidOWL possui as seguintes: projeto de ontologia em conjunto (engenheiros e especialistas no domínio), integração de informação, geração de *views* e evolução da ontologia.

O *Software Ontology Project* definiu uma metodologia ágil para construir uma ontologia com descrições formais de softwares usados na geração e análise de dados para curadoria e preservação (Copeland *et al.* , 2012). Os organizadores do projeto definiram a metodologia através de uma adaptação dos métodos ágeis da engenharia de software para a engenharia de ontologias. Os princípios fundamentais são: introdução de atividades incrementais para levantamento de requisitos e para seções de modelagem; evolução dos requisitos durante o desenvolvimento; encorajamento de times auto-organizáveis e com competências diferentes; *feedback* rápido; colaboração entre engenheiros, especialistas no domínio e usuários; lançamento de versões frequentes. A metodologia funciona através do seguinte fluxo de eventos que pode ser repetido várias vezes durante o desenvolvimento. Primeiro os requisitos são obtidos e priorizados através da prática do *planning-poker* (Grenning, 2012), em seguida, os principais requisitos são implementados e, por fim, a ontologia é avaliada.

3.6 Comparação

A ideia proposta nesse trabalho de gerar novas perguntas quando uma QC não é respondida foi inspirada no trabalho de Uschold & Gruninger (1996). Nele é apresentada a ideia de que uma questão de competência que ainda não é resolvida pelos axiomas de uma ontologia, poderia ser solucionada através da resolução de uma ou mais outras questões de competências relacionadas a QC inicial. Além disso, em seu trabalho as QCs são fundamentais na avaliação e

projeto da ontologia. A ideia de Uschold & Gruninger (1996) nunca foi implementada completamente, nem de forma automática. Portanto, criar uma implementação que sugere novas QCs para responder uma QC inicial está alinhada com o trabalho de Uschold & Gruninger (1996) para guiar engenheiros de ontologias no desenvolvimento de ontologias.

Para rastreabilidade de requisitos em engenharia de ontologia foram encontrados poucos trabalhos na literatura que abordavam essa área. A metodologia OTKM (Sure *et al.* , 2004) é um dos poucos trabalhos que consideram a rastreabilidade. Nele é apresentada uma forma simples de rastreabilidade que liga conceitos à QCs através do uso das ferramentas que foram criadas no projeto.

Como pode ser observado na Figura 3.5 o usuário consegue escrever uma QC e associar dados a ela, como quem é o engenheiro fazendo a pergunta, a data de edição, etc. Além disso, é possível executar algumas ações sobre os conceitos referenciados na pergunta. Na Figura 3.5 é mostrado um menu com opções para adicionar o conceito "*researcher*" como subclasse da raiz, adicionar uma relação com a raiz ou adicionar ele como uma instância da raiz. Com isso, a ferramenta cria ligações entre as edições feitas pelo usuário a partir dos conceitos da QC e a própria QC. Na Figura 3.6 é mostrada a recuperação das informações sobre rastreabilidade que foram gravadas. O usuário pode obter a informação de que QC está relacionada a um conceito, nesse caso ele está checando o conceito "Projekt".

A rastreabilidade de requisitos em engenharia de software é uma prática estudada e utilizada há anos, o que nos traz indícios que ela também poderia ser bem sucedida na engenharia de ontologias. Por isso, o trabalho apresentado nessa tese, desenvolve uma abordagem de rastreabilidade automática, sem interferência adicional do usuário. A rastreabilidade automática usa o processo iterativo de perguntar e responder para criar relações entre as QCs e o código OWL, além de gravar todo o histórico de interações do usuário.

Por fim, as questões de competências são amplamente usadas nas metodologias, mas, nenhuma delas traz uma abordagem de checagem de QCs de forma automática. Mesmo projetos com grandes investimentos como o NeOn citam que as questões são checadas manualmente. Automatizar esse processo pode trazer muitos ganhos para o desenvolvimento de ontologias, tanto na diminuição de erros, pois checar perguntas pode ser difícil, especialmente para axiomas inferidos, quanto na diminuição de tempo para checar se uma ontologia satisfaz todos os requisitos.

3.7 Conclusão

Neste capítulo foram apresentados diversos trabalhos que propõem metodologias e ferramentas para atividades relacionadas ao desenvolvimento de ontologias. Pode-se perceber que a engenharia de ontologias tem evoluído ao longo do tempo, mas ainda existem muitas áreas que podem ser melhoradas. A pesquisa apresentada nessa tese apresenta melhorias em áreas pouco exploradas e inovações que não são encontradas em nenhum trabalho da área.

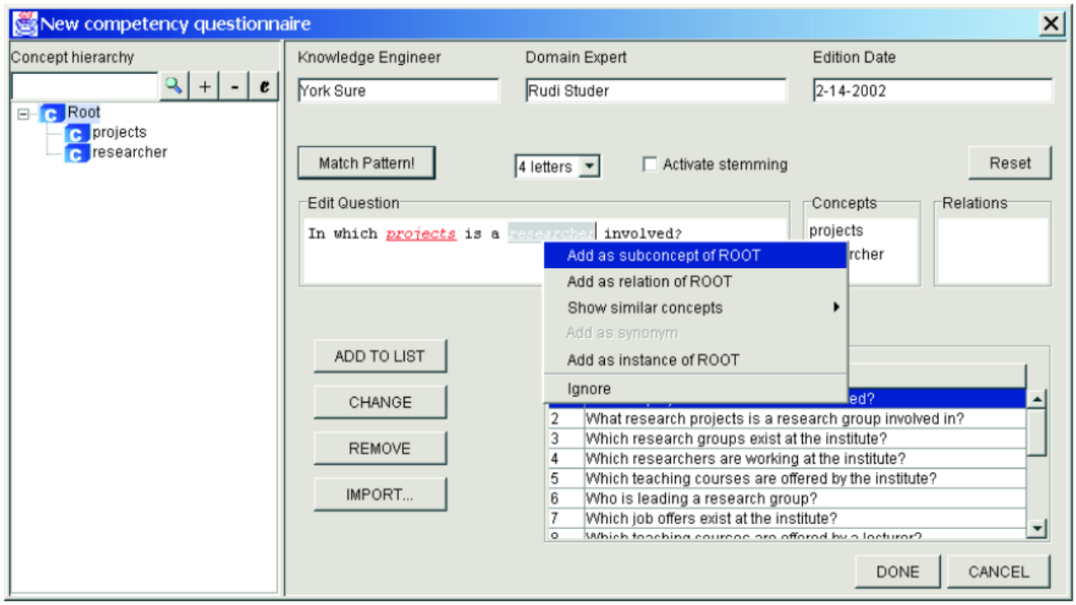

Figura 3.5: Tela da ferramenta usada no OTKM mostrando uma questão de competência

Fonte: Sure *et al.* (2002)

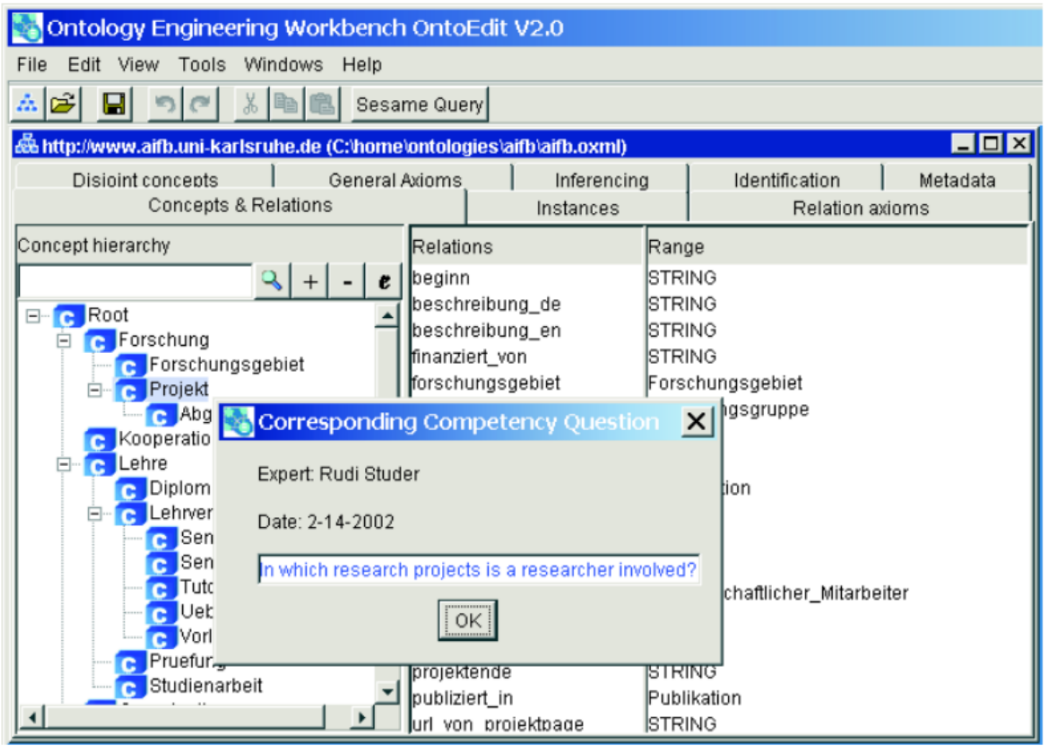

Figura 3.6: Tela da ferramenta usada no OTKM mostrando uma operação de rastreabilidade

Fonte: Sure *et al.* (2002)

Nos dois capítulos seguintes são apresentadas a proposta do método para expansão de ontologias utilizando questões de competência e os detalhes da criação da ferramenta que

implementa o método.

4

Método de Expansão de Ontologias

Este trabalho propõe um método e uma implementação (Capítulo 5) para expansão de ontologias, através de perguntas e respostas, com rastreabilidade automática de requisitos. No método, as perguntas (QCs) são analisadas automaticamente para verificar se uma ontologia possui em seus axiomas o conhecimento necessário para respondê-las corretamente. Entretanto, se a ontologia não possuir o conhecimento necessário, o método deve procurar o que está faltando para que seja possível raciocinar sobre a ontologia e retornar a resposta correta. Em outras palavras, tenta-se encontrar os axiomas que faltam para completar o conhecimento e em seguida escrevê-los na ontologia. Para isso, ao identificar as possíveis lacunas, perguntas são geradas para que o usuário responda, assim ensinando à ontologia novos conhecimentos. Por fim, após preencher as lacunas, a pergunta inicial poderá ser respondida corretamente pelo sistema.

Adicionalmente, todas as perguntas realizadas, tanto pelo usuário quanto pelo sistema, e também todas as respostas e axiomas adicionados, são armazenados. Assim, são guardadas ligações entre tudo que foi realizado através do sistema, tornando possível efetuar atividades relacionadas a rastreabilidade de requisitos para visualizar e entender a evolução do conhecimento representado na ontologia.

Considerando uma ontologia com os dois axiomas a seguir:

- $Herbivore \sqsubseteq Animal \sqcap \exists eat.Vegetable$
- $Cow \sqsubseteq Vertebrate \sqcap \exists eat.Grass$

Um usuário, utilizando o método, pode escrever a QC "*Is cow a herbivore?*" (vaca é um herbívoro?) e definir que a resposta esperada é "*true*" (verdadeiro). Isto significa que o usuário espera que o axioma $Cow \sqsubseteq Herbivore$ esteja codificado na ontologia ou que ele possa ser inferido. Um sistema implementando o método proposto tenta responder a QC do usuário, mas, com os axiomas da ontologia apresentada anteriormente, não é possível concluir que $Cow \sqsubseteq Herbivore$, pois os axiomas não definem, nem é possível inferir, tal relação. Assim, as lacunas no conhecimento precisam ser encontradas e preenchidas para que a pergunta realizada seja respondida corretamente. Então, por exemplo, as perguntas "*Is vertebrate an animal?*" (vertebrado é um animal?) e "*Is grass a vegetable?*" (grama é um vegetal?) poderiam ser

geradas e exibidas para o usuário para tentar conseguir o conhecimento necessário para responder a pergunta inicial. Se o usuário responder "*yes*" (sim) para ambas, então o sistema adiciona o conhecimento representado pelos axiomas $Vertebrate \sqsubseteq Animal$ e $Grass \sqsubseteq Vegetable$ na ontologia. Agora que as lacunas foram preenchidas, tenta-se responder a pergunta inicial novamente e dessa vez, usando a ontologia com os novos axiomas, é possível inferir $Cow \sqsubseteq Herbivore$.

Resumidamente, o método segue os passos:

1. O usuário faz uma pergunta (QC);
2. O método tenta responder a pergunta corretamente usando o conhecimento especificado na ontologia. Se conseguir, o usuário pode retornar ao passo 1 ou finalizar o uso do método, se não, segue para o passo 3;
3. Perguntas são geradas (QSs) pelo método, através da resolução de um problema de abdução, para o usuário responder;
4. O usuário fornece uma resposta para uma QS. Com isso, o conhecimento obtido é transformado em axiomas que são gravados na ontologia;
5. Volta ao passo 2.

A abordagem de perguntar ao usuário para adicionar axiomas a uma ontologia é semelhante ao método *query-the-user* apresentado por Sergot (1984). Nele, o usuário provê informações durante a execução de um programa lógico sempre que for requisitado. Além disso, usando o *query-the-user*, a interação entre o usuário e o programa é considerada simétrica, ou seja, ambos podem fazer e responder perguntas, da mesma forma que a abordagem apresentada neste trabalho. O sistema especialista APES (Hammond & Sergot, 1984) utiliza o *query-the-user* para fazer a interação em duas vias (simétrica) com o usuário.

A seguir, é apresentado em mais detalhes como o método proposto nesse trabalho é organizado, como ele está subdividido, ou seja, seus componentes, quais as principais funcionalidades desses componentes e como eles interagem entre si.

4.1 Arquitetura

Para executar os passos apresentados anteriormente, um sistema implementando o método de expansão de ontologias deve possuir quatro componentes principais: Verificador, Questionador, Construtor e Rastreador. Cada componente tem funções e responsabilidades bem definidas, trabalhando em conjunto para que o método funcione corretamente. A organização dos componentes pode ser vista na Figura 4.1. Note que os componentes têm uma ordem de execução, com exceção do Rastreador que tem acesso a todos os outros três componentes e pode ser executado

a qualquer momento. Seguindo a ordem da Figura 4.1, o Verificador é acionado primeiro, depois o Questionador e por fim o Construtor, entretanto os dois últimos só são acionados se necessário. Também é importante perceber que todos os componentes possuem acesso à ontologia que está sendo trabalhada, tanto para leitura quanto para escrita.

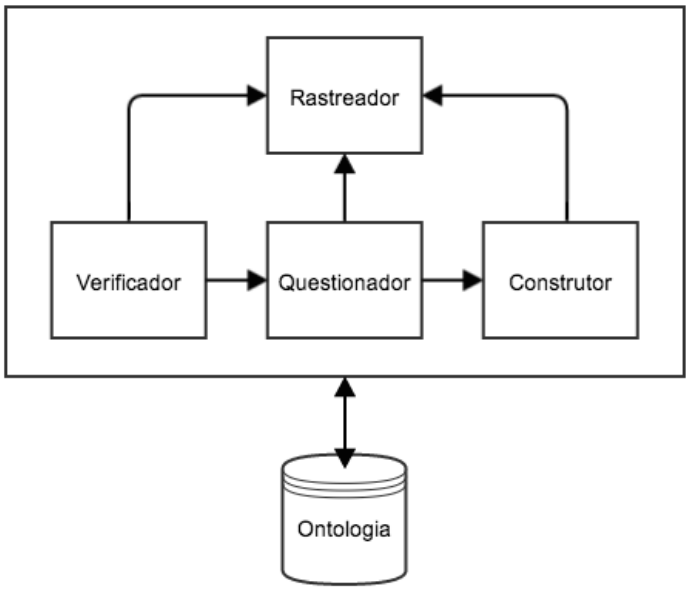

Figura 4.1: Organização dos componentes

A seguir são detalhadas as funções de cada componente e como é feita a interação entre eles.

4.2 Verificador

O componente Verificador recebe como entrada uma pergunta (QC) escrita em linguagem natural e uma resposta esperada para essa pergunta. Sua principal função é verificar se uma ontologia possui o conhecimento necessário em seus axiomas para responder uma pergunta de acordo com uma resposta esperada.

Quatro possibilidades podem ocorrer quando o componente analisa uma pergunta para buscar sua resposta:

1. A pergunta não é entendida. Isso acontece quando o usuário entra com uma pergunta e os algoritmos utilizados na implementação desse componente não conseguem definir o que está sendo perguntado. Por exemplo, uma implementação pode permitir apenas perguntas que comecem com “Do” ou “Does” em inglês, assim se o usuário fizer uma pergunta começando com “What”, o componente não conseguirá entender a pergunta. Quando isso acontecer, o componente deve retornar que a pergunta não foi entendida.

2. Os conceitos envolvidos na pergunta não estão na ontologia. Por exemplo, se for perguntado sobre carros numa ontologia sobre biologia é provável que os conceitos mencionados na pergunta sobre carros não estejam presentes na ontologia sobre biologia, fazendo com que o componente não entenda a pergunta. Quando isso acontecer, o componente deve retornar que os conceitos não foram encontrados.

3. A resposta esperada não é a mesma da resposta encontrada na ontologia. Nesse caso, o componente entende a pergunta e encontra os conceitos, mas os axiomas da ontologia não fornecem a resposta que é esperada. Quando isso acontecer, o componente deve retornar que as respostas são diferentes.

4. A resposta esperada é a mesma da resposta encontrada na ontologia. Nesse caso, o componente retorna que a ontologia possui o conhecimento para responder a pergunta com a resposta esperada.

Quando o caso número três da lista anterior acontecer, o Verificador aciona o componente Questionador, enviando para ele a pergunta realizada e a resposta esperada.

4.3 Questionador

Ao ser acionado, o componente Questionador recebe como entrada uma pergunta e a sua resposta esperada. Com isso, seu objetivo é gerar novas perguntas, que ao serem respondidas fornecerão o conhecimento necessário para que a pergunta recebida como entrada seja respondida com a resposta esperada. Como definido anteriormente, tais perguntas são chamadas de questões do sistema (QS). Após gerar a lista de QSs, através da resolução de um problema de abdução, um engenheiro deve escolher uma ou mais perguntas e respondê-las. As perguntas e respostas são passadas para o próximo componente, o Construtor.

A QS mais simples que o componente poderia gerar para obter o conhecimento necessário, seria simplesmente repetir a mesma pergunta enviada como entrada. Entretanto, esse comportamento não é recomendável, pois um dos pontos fortes desse componente é conseguir criar perguntas que ao serem respondidas conectem conhecimentos já existentes na ontologia e possivelmente gere novos além do que está sendo pedido. Gerar novas perguntas faz com que a interação entre os engenheiros e o método seja mais natural, algo próximo de um diálogo. Porém, isso nem sempre é possível, podem existir casos que o único conhecimento que pode ser adicionado para satisfazer uma QC é a própria QC.

Mais formalmente, podemos definir o problema abordado pelo Questionador como um problema de abdução em TBox (Elsenbroich *et al.* , 2006).

Definição 4.3.1. (Abdução em TBox) Dada uma ontologia O em lógica de descrição e dois conceitos C e D, sendo $O \cup \{C \sqsubseteq D\}$ consistente. A solução para o problema de abdução em TBox é um conjunto finito de axiomas Σ, tal que $O \cup \Sigma \models C \sqsubseteq D$ e $O \cup \Sigma \not\models \bot$.

No método proposto, o axioma $C \sqsubseteq D$ corresponde a uma questão de competência. Assim, dada uma QC α, no formato $C \sqsubseteq D$, que não é respondida como esperado, a abdução em TBox pode ser utilizada para encontrar um conjunto de um ou mais axiomas que quando adicionado à ontologia faz com que α seja satisfeita. Mais formalmente, tem-se, dado $O \not\models \alpha$, a abdução em TBox encontra um conjunto de axiomas Σ, tal que $O \cup \Sigma \models \alpha$.

No capítulo 5 é apresentada uma solução que utiliza unificação (Baader & Morawska, 2009) para encontrar Σ, assim definindo um conjunto de axiomas que ao serem adicionados à ontologia farão uma QC ser satisfeita.

Definição 4.3.2. (Unificação em $\mathcal{EL}$) Um problema de unificação tem a forma $\Gamma = \{C_1 \equiv^? D_1, ..., C_n \equiv^? D_n\}$, onde $C_1, D_1, ..., C_n, D_n$ são conceitos $\mathcal{EL}$. A solução ou unificador de Γ é chamada de σ, uma substituição ou conjunto de substituições da forma $\sigma(C_i) \equiv \sigma(D_i)$, para $i = 1, ..., n$. Neste caso, Γ é dito solúvel ou unificável.

Exemplo 4.3.1. (Unificação em $\mathcal{EL}$) Uma ontologia define um conceito para homem entusiasta de carros esportivos da seguinte forma: $X \equiv Human \sqcap Male \sqcap \exists loves.SportsCar$. Além disso, na ontologia existe outro conceito que representa a mesma ideia, mas de outra forma: $Y \equiv Man \sqcap \exists loves.(Car \sqcap Fast)$. Nesse caso, $X \not\equiv Y$, apesar de representarem a mesma ideia. Assim, um problema de unificação $\Gamma = \{X \equiv^? Y\}$, teria como solução a substituição: $Man \mapsto Human \sqcap Male$ e $SportsCar \mapsto Car \sqcap Fast$. Se aplicada, a substituição faria os conceitos X e Y serem equivalentes.

No problema de unificação particiona-se o conjunto N_c de nomes de conceitos, definidos na seção 2.1.1, em dois conjuntos: o primeiro, M_v, o conjunto de conceitos variáveis (isto é, conceitos que podem ser substituídos no processo de unificação) e um conjunto M_c de conceitos constantes (que não devem ser substituídos). Informalmente, diríamos que M_v é formado de nomes de conceitos que podem ter sinônimos ou que foram concebidos para serem detalhados em outra(s) descrição(ões) de conceito mais adiante.

A principal razão para o problema da unificação em $\mathcal{EL}$ estar em NP é que qualquer problema de unificação solúvel tem um unificador local, isto é, um unificador construído com átomos do problema de unificação. (Baader *et al.* , 2012).

Basicamente, qualquer problema de unificação Γ determina um número polinomial de átomos não variáveis, que são conceitos constantes ou restrições existenciais da forma $\exists r.A$, onde r é um nome de um papel e A um conceito constante ou variável. Uma atribuição S mapeia cada conceito variável X a um subconjunto S_X do conjunto de átomos não variáveis de Γ. Tal atribuição induz a seguinte relação $>_S$ sobre M_v, que é o fecho transitivo de $\{(X,Y) \in M_v \times M_v \mid Y$ ocorre em um elemento de $S_X\}$. Chamamos S de atribuição acíclica se $>_S$ é irreflexiva, ou seja nenhuma variável depende dela mesma. Qualquer atribuição acíclica S induz uma única substituição σ_S, que pode ser definida por indução sobre $>_S$ da seguinte forma:

- Se X é um elemento mínimo de M_v em relação a $>_S$, então definimos $\sigma_S(X) := \bigsqcap_{D \in S_X} D$;

- Supondo que $\sigma_S(Y)$ já está definido para todo Y tal que $X >_S Y$. Então definimos $\sigma_S(X) := \bigsqcap_{D \in S_X} \sigma_S(D)$.

Existem diferentes tipos de algoritmos desenvolvidos para unificação. O primeiro desenvolvido foi um algoritmo de força bruta "*guess and test*" (Baader & Morawska, 2009), que testa diferentes unificadores locais e verifica se eles são corretos, usando a ideia ilustrada no parágrafo anterior. Obviamente esta abordagem era extremamente ineficiente, fazendo uma busca exaustiva no espaço NP de possíveis substituições desnecessariamente. Posteriormente, foram desenvolvidos dois algoritmos mais eficientes.

O primeiro, baseado em regras, aplica diversas transformações aos conceitos do problema para chegar a uma solução. Por fim, outro algoritmo (Baader & Morawska, 2010) mapeia o problema de encontrar os unificadores para um problema de satisfatibilidade proposicional SAT (*Satisfiability Problem*) (Cook, 1971) e utiliza um resolvedor SAT (*SAT solver*).

A conexão de unificação em $\mathcal{EL}$ com a abdução em TBox (se a ontologia estiver na logica de descrição $\mathcal{EL}$) é direta, pois a solução de um problema de abdução Σ é o conjunto de axiomas que representam as transformações da unificação. Por exemplo, a substituição $X \mapsto Y$ é representada pelo axioma $X \equiv Y$.

Exemplo 4.3.2. (Unificação em EL para resolver abdução em TBox) Suponha uma ontologia O com os seguintes axiomas: $FastCar \equiv Car \sqcap \exists hasEngine.FastEngine$ e $F1Car \equiv Car \sqcap \exists hasEngine.F1Engine$. Dada a questão de competência $\alpha = FastCar \equiv F1Car$, a solução Σ tal que $O \cup \Sigma \models \alpha$, é $\Sigma = \{FastEngine \equiv F1Engine\}$.

Um algoritmo que resolva o problema de abdução em TBox pode gerar axiomas sem o que chamamos de engajamento ontológico, isto é, axiomas sem nenhuma conexão com a realidade. Por isso, o Questionador deve apresentar estes axiomas ao usuário em linguagem natural e perguntar-lhe se são verdadeiros; em caso positivo, eles devem ser inseridos na ontologia automaticamente pelo componente Construtor.

4.4 Construtor

O componente construtor recebe como entrada uma QS e uma resposta dada pelo usuário. Seu objetivo é, de acordo com a pergunta e a resposta, gerar código para ontologia, ou seja, código OWL, adicioná-lo na ontologia e por fim salvar o arquivo de especificação da ontologia com o novo conhecimento adicionado.

Após adicionar o conhecimento na ontologia, o método recomeça o processo acionando novamente o Verificador para checar se a pergunta inicialmente feita agora possui a resposta correta.

4.5 Rastreador

O componente Rastreador se liga aos outros três componentes anteriores. Ele é responsável por gravar todas as interações dos usuários utilizando o método. Ou seja, ele grava as perguntas checadas pelo Verificador, as QSs geradas pelo Questionador, as respostas fornecidas pelos usuários e o código gerado pelo Construtor. Sempre que alguma operação é realizada em um dos outros três componentes, o Rastreador é acionado.

O componente Rastreador também é responsável por fornecer operações para navegar e atuar sobre as informações armazenadas. Por exemplo, um usuário pode checar todo o histórico de uso do método para saber quais QCs foram criadas, quais respostas foram dadas, quais axiomas foram adicionados, etc. Além disso, o componente pode ser usado para analisar as relações entre as QCs, QSs, e os axiomas da ontologia. Assim, o rastreamento pode ser feito tanto dos requisitos para os axiomas e conceitos, quanto o contrário.

4.6 Trabalhos Relacionados

Poucos são os trabalhos na literatura que abordam o desenvolvimento de ontologias por completo com o foco no uso de QCs como requisitos que podem ser testados automaticamente. No trabalho de Ren *et al.* (2014) é proposta uma abordagem para facilitar a especificação e os testes dos requisitos de uma ontologia. Para isso, ela une questões de competência ao desenvolvimento orientado à testes para criar ontologias, ou seja, uma proposta semelhante ao trabalho apresentado nesta tese.

Ren *et al.* (2014) apresenta dois resultados em seu trabalho, o primeiro foi um estudo sobre como as QCs são criadas, em outras palavras, quais padrões de perguntas são mais usados por engenheiros de ontologias. O resultado encontrado possui tipos de perguntas semelhantes aos tipos de QCs suportadas pelo sistema proposto na tese. Por exemplo, tem-se *"Wh" questions*, perguntas começando com *"is"* e com *"do"*. Entre os padrões presentes em (Ren *et al.* , 2014), também pode ser encontrado um tipo de pergunta que é iniciado com *"How much"*. Essa pergunta está relacionada a quantidade, noção que não é suportada pela DL $\mathcal{EL}$, por isso não temos perguntas dessa natureza no sistema implementado.

Além de identificar os padrões de perguntas, a segunda contribuição do trabalho foi formalizar como cada tipo de pergunta pode ser testado para que seja possível saber se o requisito que elas representam está sendo satisfeito. Um diferencial da abordagem em relação ao trabalho desta tese é que a QC também é analisada para saber se ela é significativa. Por exemplo, dada a pergunta *"Which processes implement an algorithm?"*, devem ser verificado três pontos:

- As classes Process, Algorithm e o papel implements existem?
- A ontologia permite que Processes se relacionem com Algorithms através do papel implements?

- A ontologia permite que Processes não se relacionem com Algorithms através do papel implements?

Se os pontos não forem satisfeitos, o autor argumenta que não faz sentido a QC ser perguntada.

Apesar de possuir uma ideia muito próxima ao trabalho desta tese, Ren *et al.* (2014) não apresenta nenhuma implementação, sendo a sua abordagem apenas teórica por enquanto. Adicionalmente, duas inovações importantes da tese: geração de QCs e rastreabilidade automática, não são citadas no trabalho.

O trabalho de Pan *et al.* (2017) aparece como uma evolução do trabalho de Ren *et al.* (2014), mostrando o seu uso voltado para construção de *knowledge graphs*. Esse termo foi popularizado pelo Google em 2012, mas é uma ideia antiga que utiliza conceitos desde redes semânticas até as ontologias mais atuais em DL. A sua principal diferença em relação às ontologias é o seu foco em grandes quantidade de dados, sendo elas frequentemente bases muito grandes, com ferramentas para gerenciar e buscar esses dados de forma eficiente. Pan *et al.* (2017) usa as mesmas ideias e conceitos de Ren *et al.* (2014) para definir como *knowledge graphs* podem ser construídas através de uma abordagem focada em QCs que são executadas para verificar se os requisitos estão sendo satisfeitos.

Um ponto a se destacar no trabalho é a menção da importância de ferramentas para entender as consequências das operações realizadas por engenheiros, ou seja, ferramentas de rastreabilidade. Isto mostra mais uma vez que mesmo em seus estados iniciais, existem indícios da importância da rastreabilidade em engenharia de ontologias.

4.7 Conclusão

Este capítulo apresentou uma visão em alto nível da proposta desse trabalho. Nele pôde ser visto um método que propõe uma abordagem iterativa e interativa de expansão de ontologias através de perguntas e respostas para checar e para adicionar novas informações a uma base de conhecimento. Todo esse processo é feito com ênfase em operações automáticas, nas quais a interação do usuário é simples e não necessita de grande conhecimento técnico sobre os fundamentos lógicos da ontologia.

Adicionalmente foram apresentados os componentes que fazem parte do método, seus objetivos e como eles interagem entre si. Assim, tem-se uma especificação que pode ser usada para diferentes implementações. No capítulo seguinte é apresentada uma implementação do método seguindo o que foi mostrado nesse capítulo.

5

Implementação

Neste capítulo são apresentados os detalhes de uma implementação do método de expansão de ontologias introduzido no capítulo anterior, na qual os quatro componentes especificados pelo método foram implementados de acordo com suas especificações. O funcionamento interno de cada componente é detalhado nas seções posteriores desse capítulo, nas quais são mostrados os algoritmos e tecnologias utilizadas para alcançar o resultado esperado pelo método.

O sistema suporta ontologias no formato OWL que usem a linguagem $\mathcal{EL}$ da DL. Como existem grandes ontologias biomédicas codificadas nessa linguagem, então o sistema é adequado para manipulação de artefatos dessa natureza, assim como outras ontologias escritas em $\mathcal{EL}$. Além disso, em $\mathcal{EL}$ o processo de unificação utilizado no componente Questionador se torna decidível, como foi mostrado no artigo de Baader & Morawska (2009).

A implementação do sistema foi realizada utilizando a linguagem Java para as principais funcionalidades, além de HTML, CSS, JavaScript e o framework Electron[1] para a criação da interface gráfica.

A Figura 5.1 apresenta a interação dos componentes do sistema. Nela pode-se observar o componente Verificador que tem como entradas uma pergunta em linguagem natural controlada e uma resposta esperada, e como saída para o usuário, o componente retorna a avaliação da checagem. O componente Questionador recebe como entrada a pergunta e a resposta esperada do Verificador e como saída para o usuário, o componente retorna uma lista de perguntas geradas. O componente Construtor recebe como entrada uma pergunta gerada pelo Questionador e a resposta fornecida pelo usuário. Por fim, o componente Rastreador, recebe como entrada dados do Verificador, Questionador e Construtor. Respectivamente, tem-se uma pergunta e resposta esperada, uma lista de perguntas geradas e as respostas fornecidas, e um *patch* do código adicionado.

[1] http://electron.atom.io

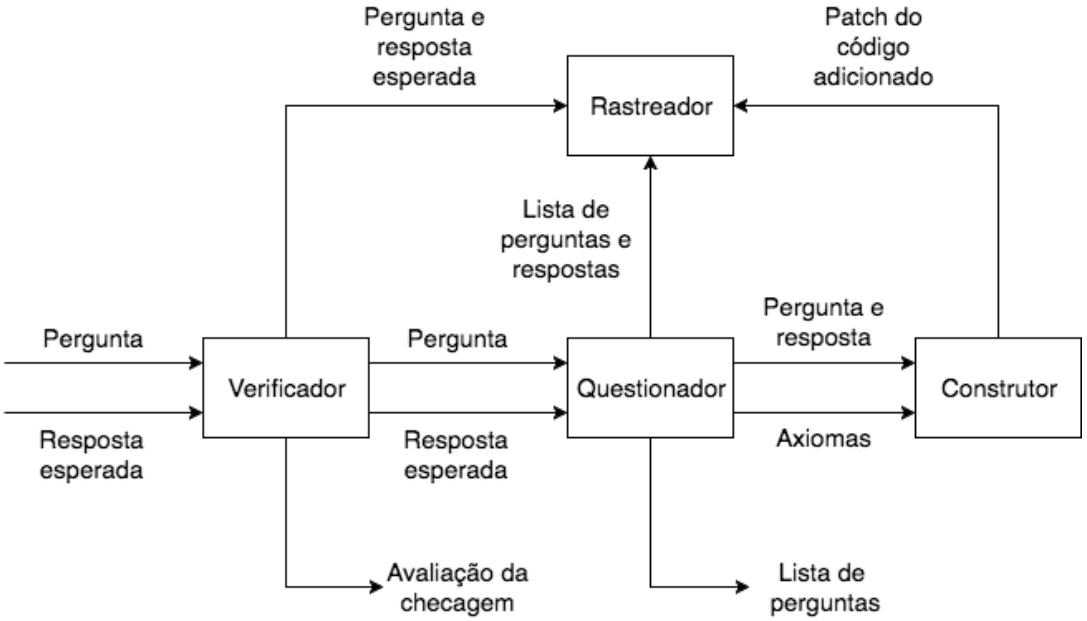

Figura 5.1: Funcionamento do Verificador

5.1 Verificador

O componente Verificador tem como principal funcionalidade receber uma QC escrita em linguagem natural controlada e sua resposta esperada, para verificar se uma ontologia possui o conhecimento necessário para responder a pergunta com a resposta passada como entrada. Nessa implementação o componente aceita como entrada questões e respostas escritas na língua inglesa.

O Verificador possui tipos pré-definidos de questões suportadas. Os tipos são especificados usando padrões, que são baseados em palavras-chave, *tokens*, *Part-of-Speech tags* e operadores de expressão regular.

Ao receber uma pergunta, o componente primeiramente divide o texto por palavras (*tokens*) e atribui para cada uma delas uma *Part-of-Speech tag* de acordo com sua função sintática. Estas operações são feitas pelo *Tokenizer*, que utiliza a biblioteca CoreNLP de Stanford (Manning *et al.* , 2014). No próximo passo, o componente avalia se os *tokens*, com as *Part-of-Speech tag*, satisfazem algum dos padrões que definem os tipos de perguntas suportados pelo Verificador. Se eles satisfizerem, uma operação para consultar o conhecimento da ontologia, de acordo com o padrão satisfeito, é realizada. Para consultar uma ontologia e conseguir extrair respostas através dos seus axiomas, o Verificador utiliza a OWL API (Horridge & Bechhofer, 2011) e o raciocinador ELK (Kazakov *et al.* , 2011), que é especializado para ontologias em DL $\mathcal{EL}$ e possui desempenho superior a diversos raciocinadores como mostrado em Kazakov *et al.* (2012). O raciocinador é importante, pois algumas respostas não estão explicitamente codificadas na ontologia, mas podem ser inferidas.

O componente suporta nove tipos de questões, que utilizam construções da gramática

da língua inglesa. Por exemplo, tem-se questões com *do* e *does*, também tem-se questões que começam com *is* e *are*, além de perguntas que começam com "*Wh*" (*what, where, who, when e which*). As perguntas foram mapeadas para cobrir operadores da DL $\mathcal{EL}$, assim tem-se perguntas que representam relações de subclasse/superclasse usando conceitos e restrições existenciais.

A Tabela 5.1 detalha cada uma delas, mostrando os padrões, os tipos de respostas que elas podem retornar e a interpretação das perguntas em DL. As letras X, Y e Z nos padrões são utilizadas para representar uma palavra qualquer que pode aparecer naquela posição. É importante deixar claro que o padrão (X)+, por exemplo, significa que uma ou mais palavras podem aparecer. Além disso, ainda tem-se os padrões com Y/Verb que significa que uma palavra precisa ser um verbo e o padrão ./Prep que especifica que qualquer preposição é aceita. As outras palavras presentes nos padrões devem ser entendidas como palavras-chave, ou seja, elas devem aparecer na pergunta da mesma forma que aparecem no padrão. O valor passado como resposta pode ser utilizado na interpretação da pergunta em DL, nos padrões esse valor é representado pela letra R. Por fim, quando uma resposta for uma lista de classes, deve-se entender a interpretação em DL como múltiplos axiomas. Por exemplo, a resposta *red wine, white wine* para uma pergunta que se encaixe no padrão **which exist** da Tabela 5.1, tem interpretação $RedWine \sqsubseteq Wine$ e $WhiteWine \sqsubseteq Wine$.

Tabela 5.1: Tipos de questões de competência suportados pelo Verificador

Is a	
Exemplo	Is red wine a wine?
Regra	is (X)+ (a|an) (Y)+
Resposta	*Yes* ou *no*
DL	$X \sqsubseteq Y$
Are	
Exemplo	Are wines beverages?
Regras	Are X Y
Resposta	*Yes* ou *no*
DL	$X \sqsubseteq Y$
Have property	
Exemplo	Does bancroft chardonnay have color some white?
Regra	(do|does) (X)+ have (Y)+ some (Z)+
Resposta	*Yes* ou *no*
DL	$X \sqsubseteq \exists hasY.Z$
Verb property	
Exemplo	Do adults drink some wine?
Regra	(do|does) (X)+ (Y/Verb) some (Z)+
Resposta	*Yes* ou *no*
DL	$X \sqsubseteq \exists Y.Z$

Preposition property	
Exemplo	Are wines made by grape?
Regra	(is|are) (X)+ Y ./Prep (Z)+
Resposta	*Yes* ou *no*
DL	$X \sqsubseteq \exists Y.Z$
Wh question	
Exemplo	What is the color of bancroft chardonnay?
Regra	(what|where|who|when) is the (X)+ of (Y)+
Resposta	Um nome representando um conceito
DL	$X \sqsubseteq \exists Y.R$
Which exist	
Exemplo	Which wines exist?
Regras	which (X)+ exist
Resposta	Uma lista de classes separadas por vírgulas ou a palavra "*and*"
DL	$R \sqsubseteq X$
Which - have property	
Exemplo	Which wines have sugar some dry?
Regras	which (X)+ have (Y)+ some (Z)+
Resposta	Uma lista de classes separadas por vírgulas ou a palavra "*and*"
DL	$R \sqsubseteq X$
	$R \sqsubseteq \exists hasY.Z$
Which - verb property	
Exemplo	Which people drink some wine?
Regras	which (X)+ (Y/Verb) some (Z)+
Resposta	Uma lista de classes separadas por vírgulas ou a palavra "*and*"
DL	$R \sqsubseteq X$
	$R \sqsubseteq \exists Y.Z$

Para responder uma pergunta, o componente precisa associar o nome dos conceitos escritos na pergunta em linguagem natural controlada aos conceitos codificados na ontologia. Como nem sempre a correspondência é direta, o componente analisa os tokens referentes a um conceito e calcula a similaridade entre esses tokens e todos os conceitos presentes na ontologia utilizando a distância de Jaro–Winkler (Winkler, 1990). O conceito mais parecido é retornado como o conceito que corresponde ao que foi escrito em linguagem natural. Assim, é possível encontrar um conceito mesmo que ele tenha sido escrito no plural, separado por espaços ou com capitalizações diferentes. Por exemplo, o texto "*red wine*" pode ser associado ao conceito "RedWine" na ontologia, e a palavra "*cows*" em uma pergunta pode ser associada à classe

"Cow". Com isso, o usuário pode escrever as questões de competência de forma natural, sem se preocupar com os detalhes da DL, nem com os detalhes da nomenclatura dos conceitos contidos na ontologia.

Dado isto, a lista a seguir mostra quando cada uma das possibilidades que podem ocorrer durante a execução do Verificador especificadas na seção 4.2 acontecem.

1. A pergunta não é entendida: acontece quando a pergunta não casa com nenhum dos padrões especificados na Tabela 5.1.

2. Os conceitos envolvidos na pergunta não estão na ontologia: esse caso não acontece na implementação, pois, mesmo que um conceito seja escrito de forma diferente do que está especificado na ontologia, a implementação procura o conceito mais parecido.

3. A resposta esperada não é a mesma da resposta encontrada na ontologia: acontece quando a resposta retornada pela consulta é diferente da resposta passada como entrada.

4. A resposta esperada é a mesma da resposta encontrada na ontologia: acontece quando a resposta retornada pela consulta é igual a resposta passada como entrada.

Para finalizar, o diagrama apresentado na Figura 5.2 sumariza o funcionamento do Verificador. Nele são mostradas as três partes principais do componente: Tokenizer, Avaliador de Padrões e Consulta, assim como as entradas e as possíveis saídas.

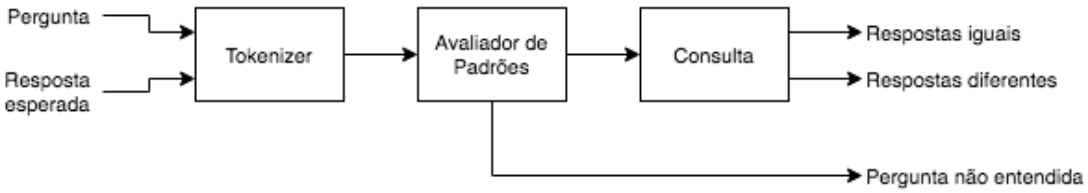

Figura 5.2: Funcionamento do Verificador

5.2 Questionador

O objetivo do Questionador é formular perguntas, que devem ser respondidas pelo usuário, para gerar novos conhecimentos que serão adicionados à ontologia, ou seja, formular as QSs. Para isso, o componente recebe uma QC e sua resposta esperada como entrada e gera, usando unificação, um conjunto de axiomas que serão transformados nas QSs. Nesse conjunto, devem estar contidos axiomas que ao serem acrescentados à ontologia adicionem o conhecimento necessário para responder a QC de entrada com a sua resposta esperada. Em seguida, os axiomas são transformados em perguntas em linguagem natural, assim concluindo a geração das QSs.

Por fim, quando respondida, a QS e a sua resposta são passadas para o componente seguinte, o Construtor. Para realizar a unificação, o Questionador utiliza a biblioteca UEL (Baader *et al.* , 2012) implementada em Java e com código disponível na web[2].

O componente funciona em cinco etapas:

1. Definição da equação do problema de unificação;
2. Definição dos axiomas que são considerados na unificação;
3. Definição das variáveis;
4. Unificação;
5. Conversão para linguagem natural.

Cada etapa é apresenta em detalhes a seguir.

5.2.1 Definição da Equação

Um problema de unificação é um conjunto de equações da seguinte forma: $\Gamma = \{C_1 \equiv^? D_1, ..., C_n \equiv^? D_n\}$. Para o Questionador, o problema de unificação sempre será um conjunto com uma equação, ou seja, $\Gamma = \{C_1 \equiv^? D_1\}$, que corresponde a QC e a resposta esperada passadas como entrada.

Além de axiomas de equivalência, a biblioteca UEL também suporta axiomas de subclasse como equação do problema de unificação. Por isso, para definir a equação, o componente considera a interpretação em DL da QC e sua resposta. Assim, a equação é definida de acordo com a Tabela 5.1 apresentada anteriormente. Por exemplo, dada a pergunta "Does player have car some ferrari", com a interpretação em DL $Player \sqsubseteq \exists haveCar.Ferrari$, o problema de unificação seria $\Gamma = \{Player \sqsubseteq^? \exists haveCar.Ferrari\}$

Concluindo a primeira etapa do componente, a equação é guardada para ser utilizada na unificação.

5.2.2 Definição dos Axiomas

Além da equação, é necessário definir os axiomas que serão considerados na unificação. Para o sistema, todos os axiomas da ontologia são considerados, assim o algoritmo de unificação analisa todos os conceitos e pode efetuar unificações utilizando todo o conhecimento representado na ontologia.

Ao finalizar essa etapa, um conjunto de axiomas é definido para ser utilizado na etapa da unificação.

[2]https://github.com/julianmendez/uel

5.2.3 Definição das Variáveis

A última definição necessária antes da unificação é a do conjunto de variáveis (M_v), ou seja, o conjunto de conceitos que podem ser substituídos durante a unificação. Para isso, é utilizada a mesma abordagem que o plugin da UEL para o Protégé utiliza para sugerir para o usuário os conceitos que podem ser variáveis. O plugin é disponibilizado junto com a biblioteca usada no sistema.

A abordagem do plugin da UEL começa armazenando a equação definida. Em seguida, todos os axiomas que definem os conceitos presentes na equação são guardados. Continuando, são armazenados todos os axiomas que definem os conceitos presentes nos axiomas que foram guardados anteriormente. O processo continua de forma recursiva até que não existam mais definições de conceitos presentes nos axiomas guardados.

Por exemplo, dada a equação $X \equiv Y$ e dado o conjunto de axiomas:

- $X \equiv A$
- $Y \equiv B$
- $B \equiv C$
- $D \equiv E$

Primeiro é guardado o axioma $X \equiv Y$. A seguir, são guardados os axiomas que definem conceitos presentes na equação, nesse caso tem-se $X \equiv A$ e $Y \equiv B$. Recursivamente, agora são guardados os axiomas que definem os conceitos presentes nos dois axiomas mostrados anteriormente, ou seja, o axioma $B \equiv C$. O processo encerra, pois não existem mais definições de conceitos presentes nos axiomas guardados. Note que o axioma $D \equiv E$ não foi incluído, pois ele não traz uma definição de um conceito utilizado na equação, nem nos axiomas armazenados recursivamente.

O passo seguinte é, dados os axiomas, escolher os conceitos que podem ser variáveis. Para isso, são adicionados a um conjunto ψ_{M_v} todos os conceitos presentes nos axiomas armazenados que não possuem definição. Continuando com o exemplo anterior, os axiomas armazenados foram: $\{X \equiv Y, X \equiv A, Y \equiv B, B \equiv C\}$, então os conceitos presentes nesses axiomas são: $\{X, Y, A, B, C\}$, mas dos conceitos apresentados, os que não possuem definições são: $\psi_{M_v} = \{A, C\}$.

Além de considerar cada classe de ψ_{M_v} individualmente como variável, o componente também considera como possíveis variáveis os elementos resultantes da combinação dois a dois do conjunto ψ_{M_v}, que chamaremos de $C\psi_{M_v}$ Isso é feito, pois uma unificação pode ser realizada com mais de uma variável, ou seja, efetuando substituições em mais de um conceito.

Ao fim dessa etapa, as variáveis são definidas e guardadas para serem utilizadas na unificação.

5.2.4 Unificação

A quarta etapa do componente realiza a unificação da equação definida na primeira etapa de acordo com os axiomas e as variáveis definidos na segunda e terceira etapa respectivamente.

Para cada elemento de ψ_{M_v} e para cada par de $C\psi_{M_v}$, a UEL é utilizada para computar os unificadores. Assim, gerando uma lista de transformações que, ao serem efetuadas, farão a equação do problema de unificação ser satisfeita. No final da etapa, todos os unificadores gerados são armazenados para serem utilizados na etapa de conversão para linguagem natural.

5.2.5 Conversão para Linguagem Natural

Para finalizar, o componente precisa executar mais uma etapa para transformar cada unificador gerado em linguagem natural. Dessa forma, o componente retorna para o usuário perguntas, as QSs, que podem ser entendidas mais facilmente e respondidas para gerar conhecimento.

Os unificadores gerados na etapa anterior têm o formato $X \equiv C$, sendo C um conceito complexo. O algoritmo inicia a conversão em linguagem natural, adicionando C a uma fila e inicializando uma flag com valor vazio. A partir desse passo inicial, o primeiro item da fila é retirado e é efetuada uma série de testes para saber qual o seu tipo, ou seja, se ele é uma classe, um papel, uma restrição existencial ou uma conjunção. De acordo com o tipo encontrado e com o valor da flag, o componente vai construindo a pergunta passo a passo. Se o item da fila for uma restrição existencial ou uma conjunção, o item é quebrado em itens menores e eles são adicionados a fila. Por outro lado, se o item da fila for uma classe ou um papel, o componente retorna texto que faz parte da pergunta. Esse processo se repete até que a fila esteja vazia.

Os nomes das classes e papéis presentes nas perguntas são tratados para que eles tenham um formato natural. O componente adiciona um espaço em branco antes de cada letra maiúscula, exceto a primeira, e no final converte todo o nome para letras minúsculas. Assim, uma classe chamada “ProcedureOnSkeletalSystem” aparece na pergunta como “procedure on skeletal system”. Já um papel com o nome “findingSite” aparece como “finding site”.

A Figura 5.3 ilustra todos os passos do algoritmo de geração das perguntas. É importante notar, que a pergunta vai sendo construída passo a passo, então quando o fluxograma chega num dos itens de retorno, os itens que contém um string (texto entre aspas), esse texto é concatenado com o que já foi gerado anteriormente.

5.2.6 Exemplo

Nessa seção é apresentado um exemplo passo a passo do funcionamento do componente Questionador. Considere uma ontologia com os seguintes axiomas:

- $FastEngine \sqsubseteq CarEngine$
- $RaceEngine \sqsubseteq CarEngine$

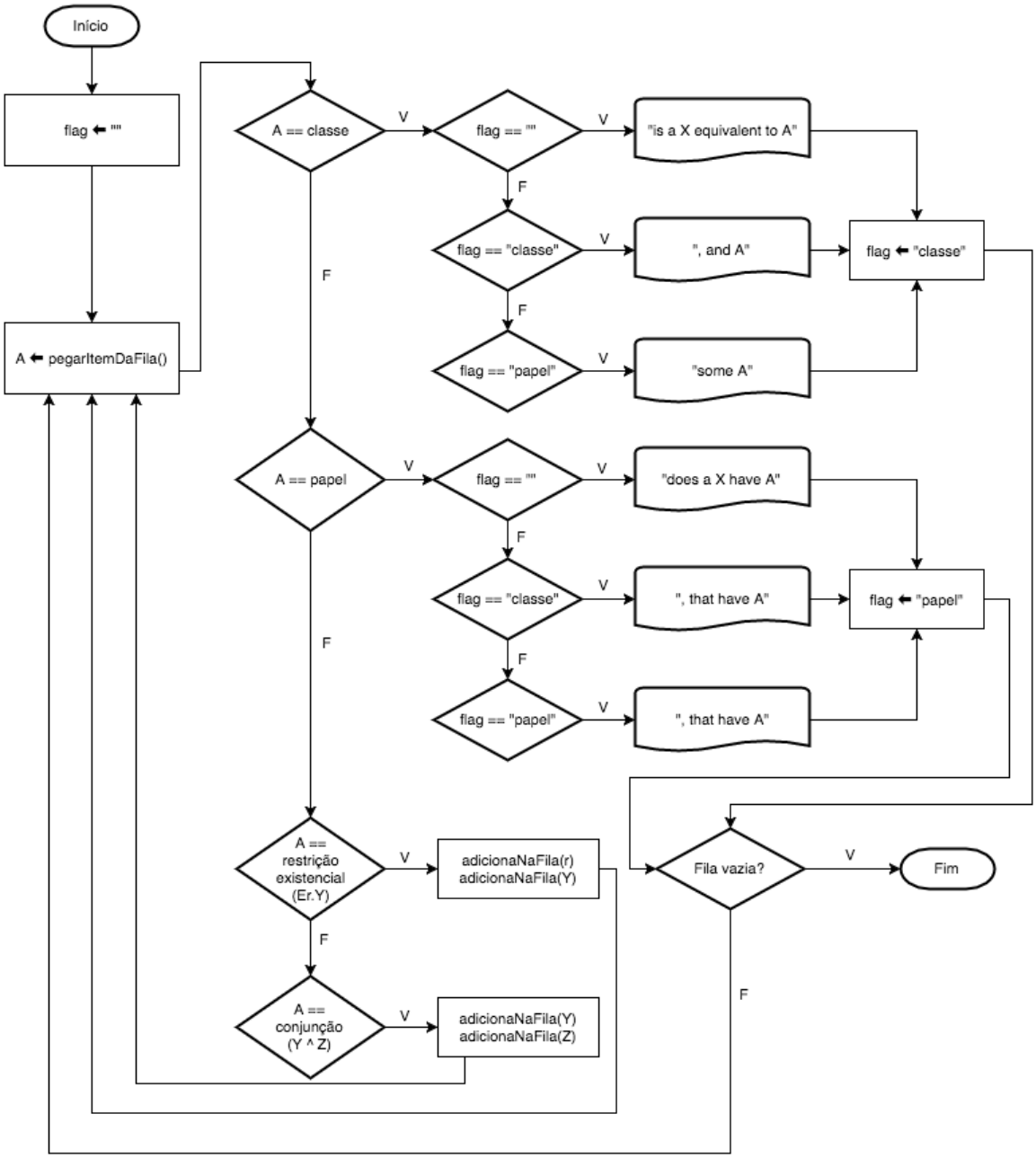

Figura 5.3: Fluxograma do algoritmo de conversão para linguagem natural

- $Car \sqsubseteq Vehicle$
- $FastCar \sqsubseteq Car \sqcap \exists engine.FastEngine$
- $RaceCar \sqsubseteq Car \sqcap \exists engine.RaceEngine$

O algoritmo é iniciado recebendo como entrada a QC "is race car a fast car?" com a resposta esperada "true". Na primeira etapa é definida a equação utilizada no problema de unificação. Para a pergunta de entrada, a equação é: $RaceCar \sqsubseteq FastCar$. Na segunda etapa, os axiomas a serem utilizados são definidos. No componente Questionador, todos os axiomas da ontologia são usados na unificação.

As variáveis são definidas na terceira etapa. Para isso a equação é armazenada, assim como os axiomas que definem conceitos presentes na equação e os axiomas que definem os conceitos presentes nesses novos axiomas de forma recursiva. Portanto, para esse exemplo, temos que os os axiomas armazenados são:

- $RaceCar \sqsubseteq FastCar$
- $FastCar \sqsubseteq Car \sqcap \exists engine.FastEngine$
- $RaceCar \sqsubseteq Car \sqcap \exists engine.RaceEngine$

Em seguida, todos os conceitos presentes nos axiomas armazenados que não possuem definições são adicionados a ψ_{M_v}. Assim, tem-se $\psi_{M_v} = \{Car, FastEngine, RaceEngine\}$. Para finalizar, a combinação dois a dois de ψ_{M_v} é calculada, $C\psi_{M_v} = \{(Car, FastEngine), (Car, RaceEngine), (FastEngine, RaceEngine)\}$. Na quarta etapa a unificação é realizada pela biblioteca UEL. Quando a unificação é feita considerando a variável *Car*, o unificador resultante é $Car \equiv \exists engine.FastEngine$. Já considerando a variável *RaceEngine*, o unificador resultante é $RaceEngine \equiv FastEngine$. Para as outras variáveis não é possível encontrar unificadores válidos.

Por fim, os dois unificadores encontrados são transformados em texto em linguagem natural. Para o unificador $Car \equiv \exists engine.FastEngine$, a conversão começa adicionando $\exists engine.FastEngine$ na fila e iniciando a flag como vazia. O próximo passo é retirar o primeiro item da fila e testar qual o seu tipo, se é uma classe, um papel, uma restrição existencial ou uma conjunção. Nesse caso, tem-se uma restrição existencial, então o papel *engine* e a classe *FastEngine* são adicionados na fila nessa ordem. O algoritmo volta para pegar novamente um item da fila, que dessa vez é o papel *engine*. Assim, após testar o tipo do item da fila e como a flag está vazia, o algoritmo retorna a primeira parte da pergunta, que é "does a car have engine" e atribui o valor "papel" à flag. Como a fila ainda não está vazia, então o próximo item é retirado. O item *FastEngine* é uma classe e a flag atual tem o valor "papel", então o texto retornado é apenas o nome da classe, nesse caso "fast engine", e a flag passa a ter o valor "classe". Como, *FastEngine* era o último item da lista, então o algoritmo termina sua execução e o unificador em forma de pergunta em linguagem natural é: "does a car have engine some fast engine?".

Para o segundo unificador encontrado, $RaceEngine \equiv FastEngine$, o algoritmo começa adicionando *FastEngine* à fila e iniciando a flag como vazia. Agora tem-se um caso mais simples, pois não existem conjunções nem restrições existenciais. Após pegar o primeiro item da fila, como *FastEngine* é uma classe e a flag é vazia, o algoritmo retorna o texto "is a race engine equivalent to fast engine". Ao testar se a fila está vazia, o algoritmo recebe uma resposta verdadeira e termina sua execução. Assim, tem-se que que o unificador em forma de pergunta em linguagem natural é: "is a race engine equivalent to fast engine?".

5.3 Construtor

O componente Construtor é acionado quando o usuário responde uma QS gerada pelo componente Questionador. O seu objetivo é transformar uma pergunta e sua resposta em código OWL e adicioná-lo em uma ontologia

O Construtor utiliza a OWL API para manipular os axiomas, pois ela consegue trabalhar de forma transparente para o desenvolvedor com diferentes representações de axiomas, sejam eles em DL ou em OWL. Quando o Questionador termina de gerar as QSs, ele passa para o Construtor, o texto em linguagem natural e os axiomas relacionados a QS. Assim, o Construtor utiliza a OWL API para tratar diretamente os axiomas como código OWL e escrevê-los na ontologia que está sendo trabalhada.

5.4 Rastreador

Todas as interações realizadas através do sistema são rastreáveis e é responsabilidade desse componente prover os meios para que isso aconteça. Além disso, o Rastreador também fornece operações para navegar e atuar sobre esses dados.

Quando um usuário checa uma QC através do Verificador, o Rastreador armazena a QC. Se o Questionador gerar novas perguntas, pois não foi possível responder a pergunta inicial, o Rastreador armazena as QSs e a relação entre elas e a pergunta inicial que não foi respondida. Quando o usuário responde uma QS, o Rastreador armazena a resposta, cria a relação entre a resposta e a QS e também armazena um *patch* do código OWL que foi adicionado a ontologia. O *patch* é gerado através de uma operação de *diff* usando o algoritmo de Myers (1986). Dessa forma, todos os passos do processo de expansão de uma ontologia são monitorados e podem ser consultados para análise futura.

Na Figura 5.4 pode ser vista a estrutura de dados do Rastreador. Onde QC_i representa as questões de competência e i é o índice da QC que pode variar de 1 até n. Por exemplo, QC_1 é a primeira QC checada, QC_2 a segunda, etc. QS_{ij} representa as questões formuladas pelo sistema, onde i é o índice da QC que a QS está relacionada e j é o índice da QS, este último varia de 1 até m. Por exemplo, QS_{12} é a segunda QS relacionada à primeira QC e QS_{31} é a primeira QS relacionada à terceira QC. Por fim, R_{ij} é a resposta do usuário para a QS_{ij} e P_{ij} é o *patch* do código adicionado de acordo com a resposta R_{ij} da QS_{ij}. Essa estrutura de dados é serializada em um arquivo e sempre que necessário a estrutura pode ser lida ou modificada pelo componente.

Usando os dados armazenados na estrutura de dados apresentada, o Rastreador fornece as seguintes operações:

Log: essa operação exibe todas as QCs, QSs e respostas armazenadas em uma hierarquia. Assim, o engenheiro de ontologias possui uma maneira de obter uma visão geral de tudo que foi gerado até o momento da execução da operação. Ou seja, ele pode observar quais as QCs

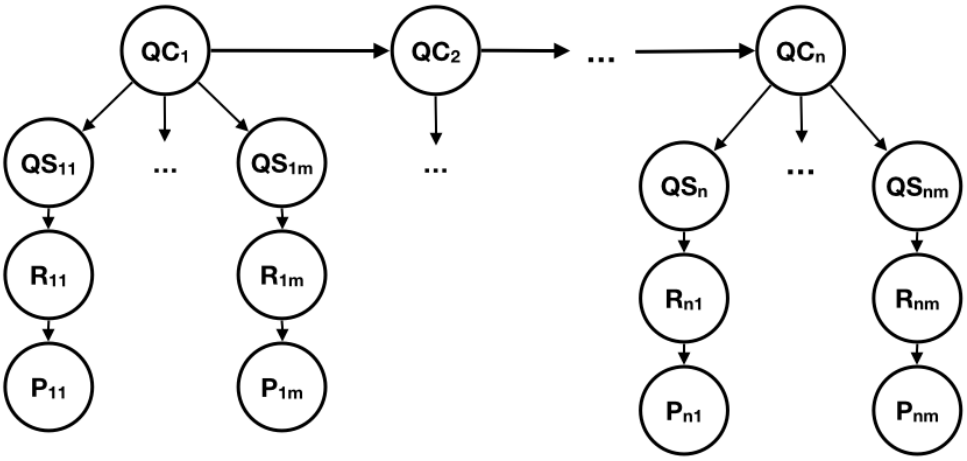

Figura 5.4: Estrutura de dados do Rastreador

foram criadas e em que ordem foram criadas, se foram formuladas QSs para as QCs e, se foram, quais as QSs e suas respostas. Para gerar a exibição, o componente percorre todos os nós da estrutura de dados e mostra as informações deles para o usuário.

Diff: usando essa operação, o usuário especifica uma QS armazenada e o sistema exibe o código OWL que foi adicionado ou removido quando essa QS foi respondida. Isso é possível, pois o componente salva o *patch* com informações do código adicionado ou removido, ou seja, apenas as linhas de código que foram modificadas e onde elas foram modificadas no arquivo da ontologia. Assim, o componente tem todos os dados que ele precisa para exibir o *diff* para o usuário.

Coverage: essa operação pode ser usada para QCs e QSs. Através dela o usuário recebe como retorno todos os conceitos que estão presentes na QC ou QS e todos os axiomas na ontologia que usam esses conceitos. Assim, tem-se uma visão de quais conceitos e axiomas estão relacionados e são impactados pelas perguntas realizadas. A busca pelos conceitos e pelos axiomas que usam os conceitos é feita através da OWL API.

Class-question: com essa operação o usuário especifica uma classe e recebe como retorno quais QCs ou QSs referenciam essa classe. Para isso, o componente percorre a estrutura de dados gravada procurando as perguntas que referenciem a classe de entrada. Essa operação traz o caminho inverso da operação Coverage, fazendo com que o componente Rastreador consiga seguir os rastros tanto no sentido das perguntas para os axiomas e classes (Coverage) quanto no sentido das classes para as perguntas (Class-question).

Rollback: se o usuário realizou mudanças que não são mais desejadas, o sistema permite ele desfazer alterações. A operação de *rollback* retorna o processo de expansão da ontologia para um ponto específico. Este ponto é determinado especificando QCs e QSs no histórico de mudanças apresentado pela operação de *log*. Novamente, isso é possível, pois o sistema

salva o *patch* do código modificado. Então, para desfazer as mudanças, o sistema remove a aplicação de cada *patch* desde o estado atual até, mas não incluindo, o *patch* da QS especificada.

5.5 Usando a Ferramenta

Nessa seção é apresentada a interface gráfica criada para o sistema implementado. Além disso, mostraremos um passo a passo do uso de todas as funcionalidades disponíveis na ferramenta, exibindo suas telas correspondentes com os resultados alcançados.

A tela inicial é apresentada na Figura 5.5. Na parte superior da janela estão presentes três botões: *Open*, *Check* e *Track*. O primeiro serve para abrir uma ontologia no sistema, o segundo para alternar para a tela de checagem da ontologia, que é a tela inicial do sistema, e o terceiro para alternar para a tela de rastreamento da ontologia. Logo abaixo, tem-se dois campos de texto seguidos do botão *Ask*. Da esquerda para direita, o primeiro campo recebe uma questão de competência a ser checada e o segundo campo a resposta esperada para a QC. O botão *Ask* serve para realizar a pergunta usando o sistema. No centro da janela, um texto para auxiliar o usuário do sistema é exibido, mas ao ser efetuada uma pergunta, esse texto desaparece, dando lugar às informações relacionadas a checagem das QCs. Por fim, ainda na tela principal, no canto inferior direito, existe uma opção para abrir uma ajuda, que mostra os tipos de questões suportadas, sua sintaxe, sua relação com lógica de descrição e exemplos de uso das perguntas.

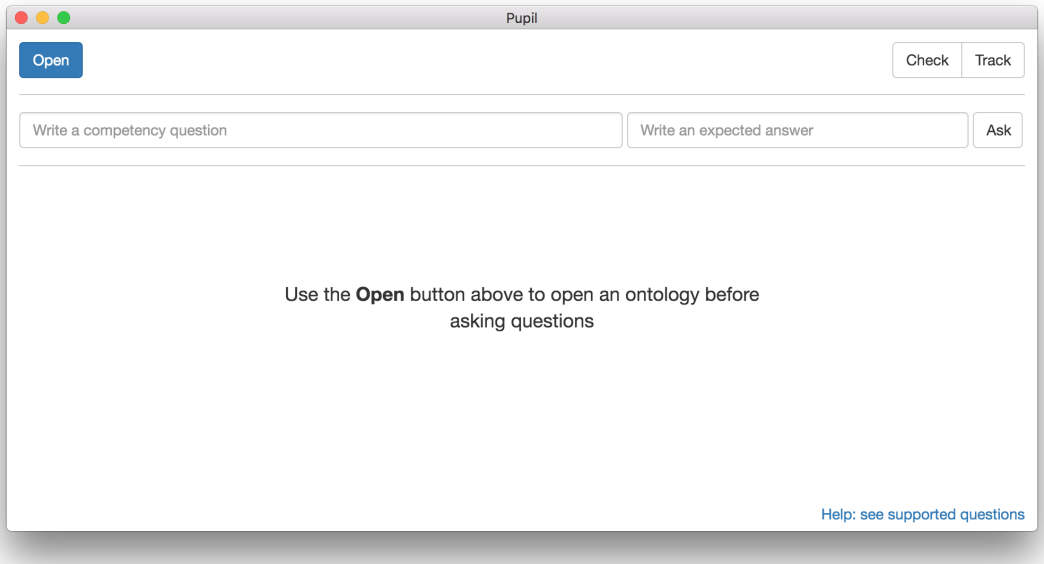

Figura 5.5: Tela inicial da ferramenta

Para iniciar o uso da ferramenta, aciona-se o botão *Open* para escolher uma ontologia. No passo a passo aqui apresentado, usaremos uma ontologia com os seguintes axiomas:

- $RedWine \equiv Wine \sqcap \exists color.Red$
- $FooWine \equiv Wine \sqcap \exists color.Foo$

Com a ontologia carregada, a ferramenta passa a exibir o caminho para o arquivo da ontologia ao lado do botão *Open*, como pode ser observado na Figura 5.6.

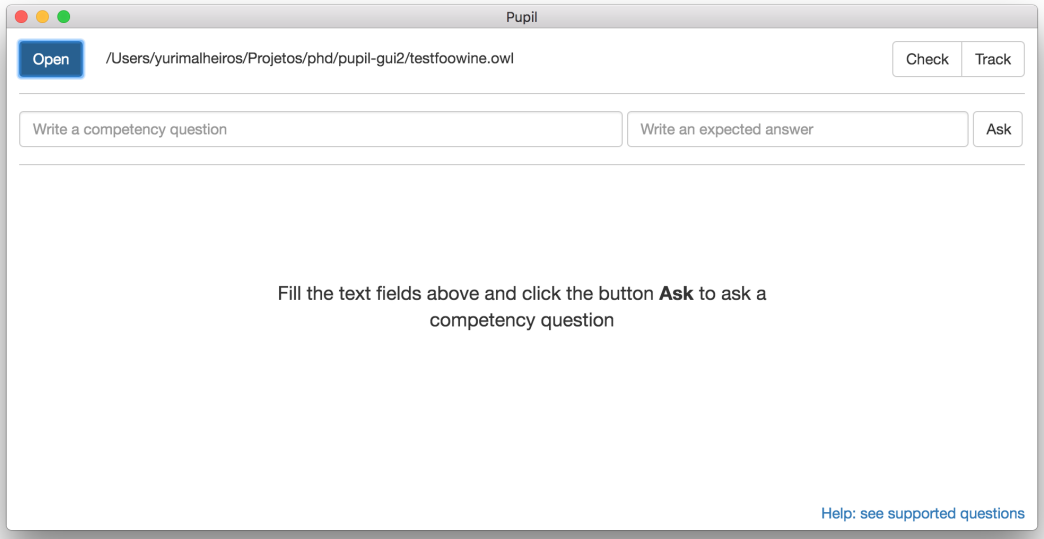

Figura 5.6: Tela da ferramenta após carregar a ontologia testfoowine.owl

O próximo passo no uso da ferramenta é a escrita de questões de competência. Para isso, basta digitar a pergunta no primeiro campo de texto, a resposta esperada no segundo campo de texto e clicar no botão *Ask* para que a ferramenta processe a pergunta e retorne uma resposta. A Figura 5.7 mostra o resultado apresentado para o usuário após ser feita a pergunta: “is red wine a wine?”. Nela pode-se ver a pergunta realizada, a sua resposta dada de acordo com os axiomas da ontologia, e, entre parênteses, a ferramenta exibe a interpretação em DL da pergunta. Na pergunta exibida na Figura 5.7 foram escritos “red wine” e “wine” para representar os conceitos desejados, mas tais conceitos na ontologia se chamam respectivamente “RedWine” e “Wine”. Como o sistema procura por conceitos similares aos conceitos escritos na pergunta, a exibição do axioma em DL deixa claro para o usuário que conceitos estão sendo considerados e qual a relação entre eles na pergunta.

Na Figura 5.8 é mostrado o resultado para outra QC, nesse caso “is foo wine a red wine?”. Como a classe FooWine não possui relação de subclasse com RedWine, a QC não é respondida positivamente. Quando isso acontece a ferramenta aciona o Questionador para gerar novas perguntas. Note que na Figura 5.8, além da informação sobre a pergunta realizada e a sua resposta, também pode-se ver uma lista de novas perguntas, que são as QSs.

Para responder uma QS basta clicar na questão que um campo de texto é apresentado para o usuário digitar a resposta. Ao pressionar *enter* a ferramenta aciona o Construtor, que

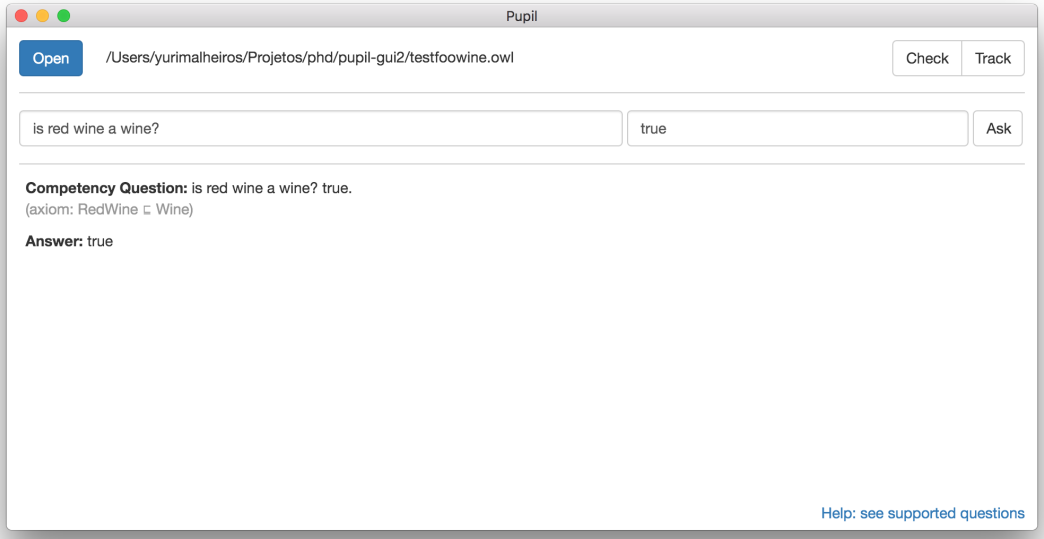

Figura 5.7: Tela da ferramenta com a checagem da questão de competência "is red wine a wine?"

adiciona o conhecimento na ontologia, e ela checa novamente a QC perguntada inicialmente para saber se ela agora é satisfeita. Na Figura 5.9 foi selecionada a pergunta "is foo equivalent to red?" e a resposta dada foi "yes". Já na Figura 5.10 é mostrada nova checagem após a adição dos axiomas à ontologia.

Para utilizar as funções de rastreamento é necessário acionar o botão *Track* no canto superior direito. Como pode ser observado na Figura 5.11, a tela de rastreamento é dividida em três partes. Na esquerda é mostrada a hierarquia de classes da ontologia carregada. No meio é apresentado o *log* das operações realizadas com a ferramenta, nesse caso, as duas QCs perguntadas, a QS e a resposta dada pelo usuário. E, na parte da direita, tem-se um espaço em branco que é utilizado para exibir informações relacionadas as QCs e QSs.

Ao clicar numa QC a ferramenta exibe as informações trazidas pela operação Coverage do Rastreador. A Figura 5.12 apresenta a tela quando o usuário clica na QC "is red wine a wine?". Nela podem ser vistos do lado direito os conceitos presentes na pergunta e os axiomas que usam esses conceitos na ontologia.

Quando o usuário clica numa QS também é mostrado o resultado da operação Coverage, mas, além disso, a ferramenta também exibe um botão com o texto "show code diff" e o axioma em DL adicionado quando a QS foi respondida. A Figura 5.13 mostra a tela da ferramenta após o usuário clicar na QS "is foo equivalent to red?". Ao clicar no botão "show code diff", a ferramenta exibe quais linhas de código foram modificadas no arquivo OWL quando o novo conhecimento foi adicionado à ontologia. Isto pode ser visto na Figura 5.14. A indicação @line: <número> especifica que as linhas adicionadas exibidas abaixo começam na linha do número mostrado. Assim, nesse exemplo, tem-se que foram adicionadas duas linhas a partir da linha 36

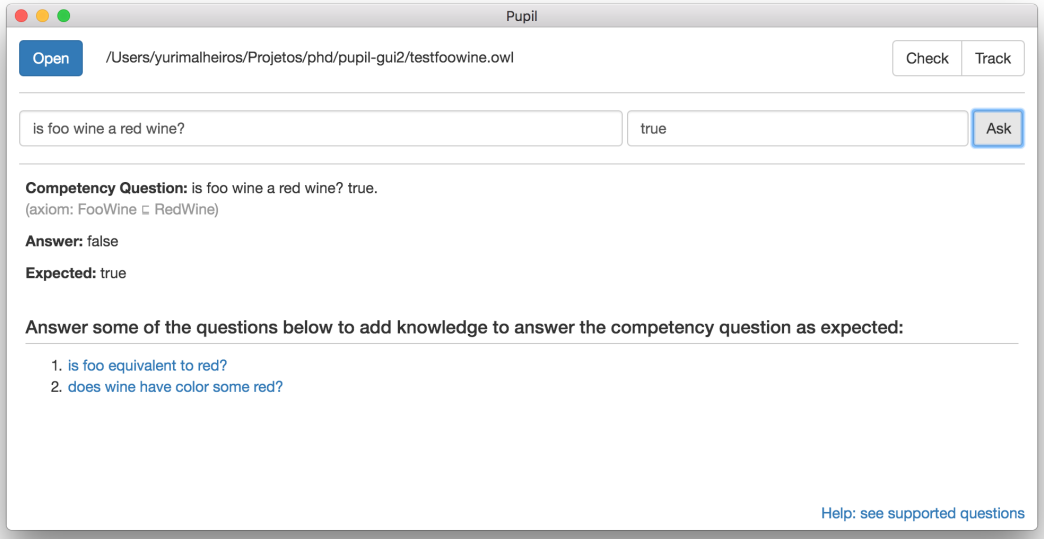

Figura 5.8: Tela da ferramenta com a checagem da questão de competência "is foo wine a red wine?"

do arquivo OWL, uma linha em branco a partir da linha 39, mais duas linhas a partir da linha 42 e por fim uma nova linha em branco a partir da linha 45.

Na hierarquia de classes mostradas no lado esquerdo da tela de rastreamento é possível clicar nos nomes das classes. Quando essa ação é executada, a ferramenta chama a operação Class-question do Rastreador para saber quais perguntas referenciam a classe clicada. Como pode ser visto na Figura 5.15, a ferramenta destaca com uma borda vermelha as QCs ou QSs que contenham a classe selecionada na hierarquia, nesse caso, a classe RedWine.

5.6 Conclusão

Este capítulo apresentou todos os aspectos da implementação do método de construção de ontologias através de perguntas e respostas com um sistema de rastreabilidade automática de requisitos. Cada componente teve seu funcionamento explicado, mostrando os algoritmos e tecnologias usadas para criar uma ferramenta que segue o método proposto no capítulo anterior. Por fim, foi mostrado um passo a passo do uso das funcionalidades da ferramenta através de sua interface gráfica.

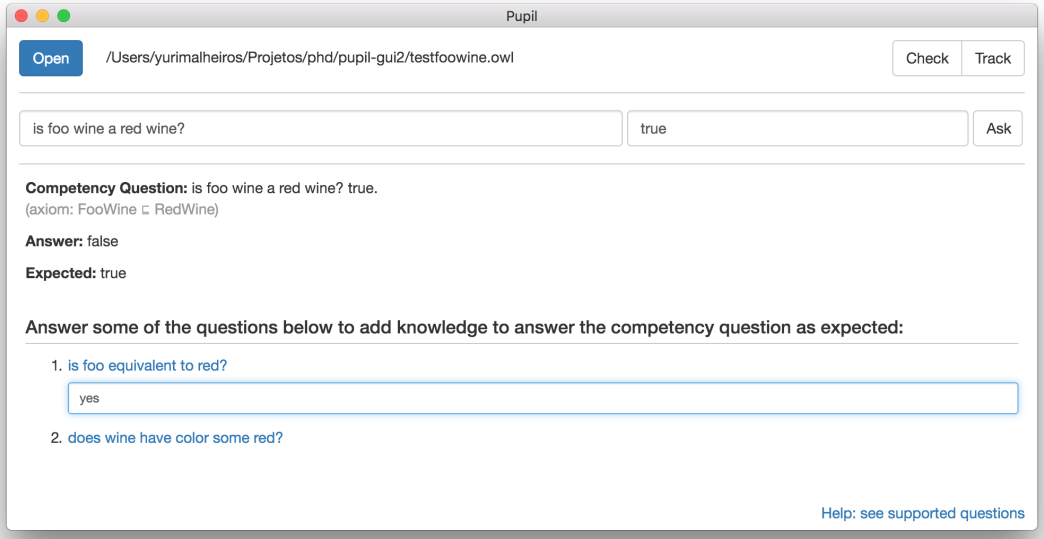

Figura 5.9: Tela da ferramenta com a resposta a uma questão criada pelo sistema

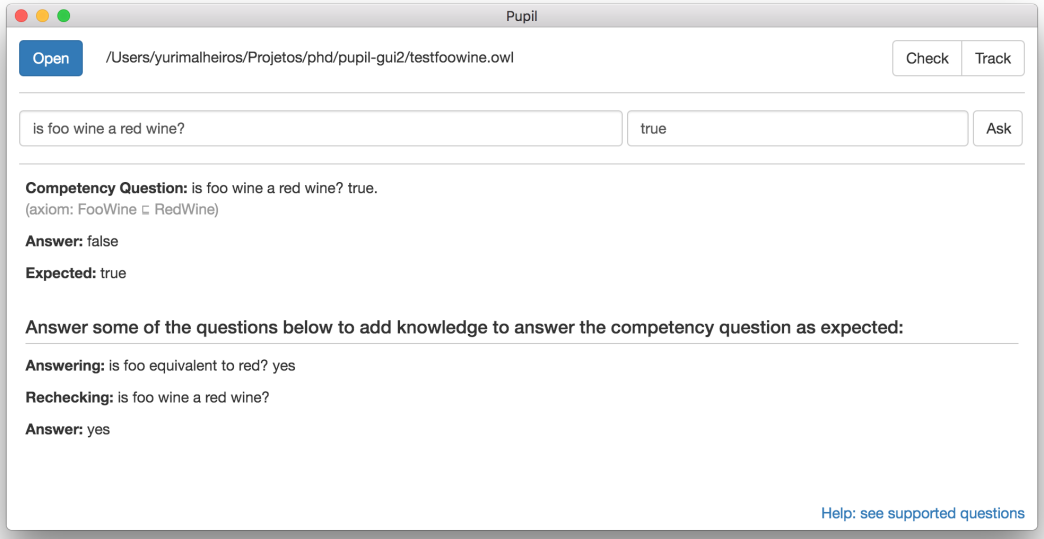

Figura 5.10: Tela da ferramenta com a resposta e a nova checagem da QC inicial

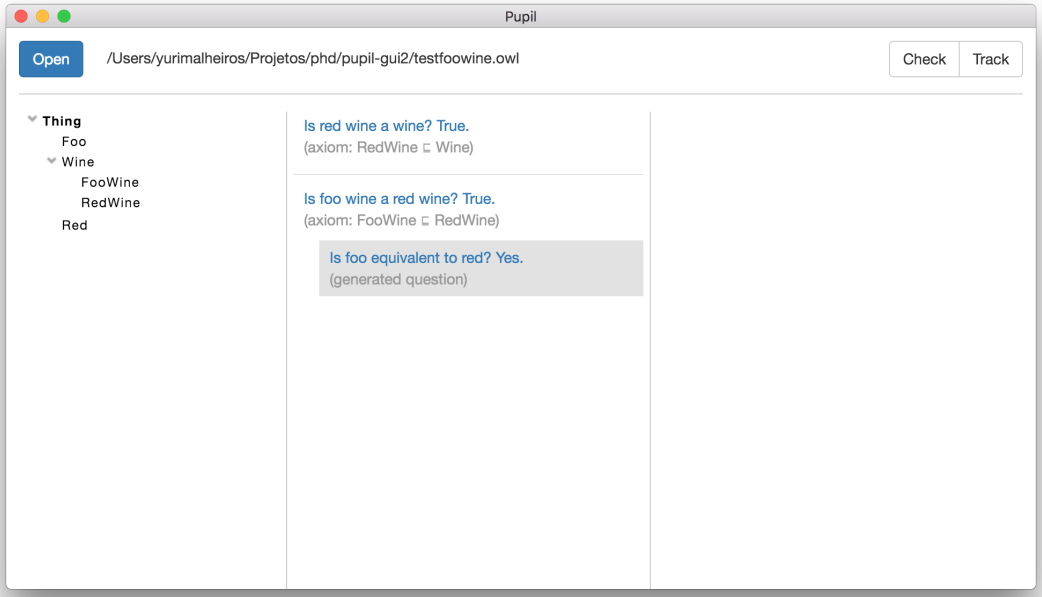

Figura 5.11: Tela da ferramenta mostrando a hierarquia de classes da ontologia e o *log* das operações realizadas

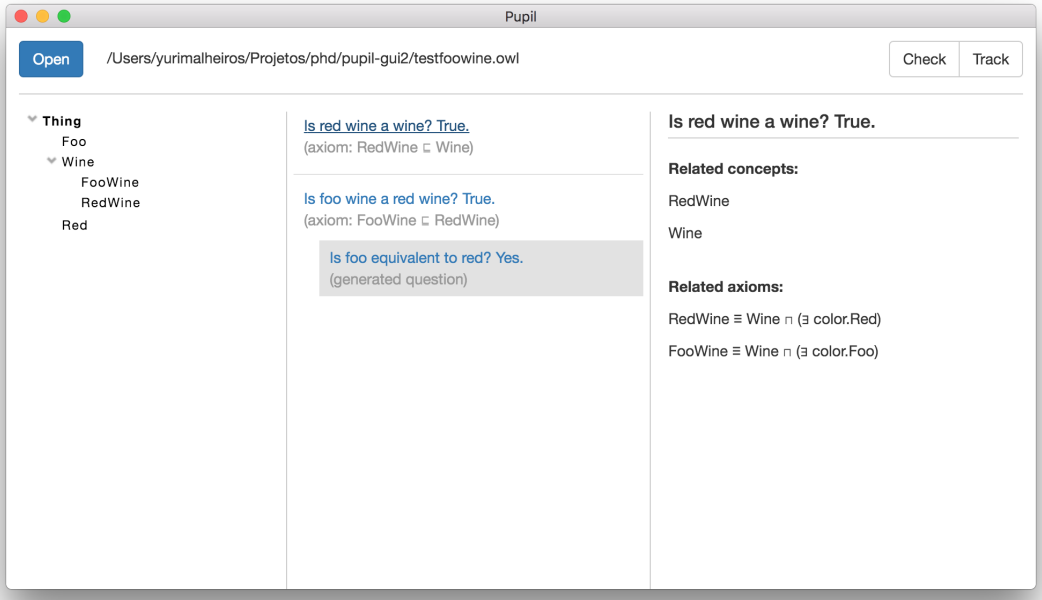

Figura 5.12: Tela da ferramenta mostrando o resultado da operação *coverage* para a QC "is red wine a wine?"

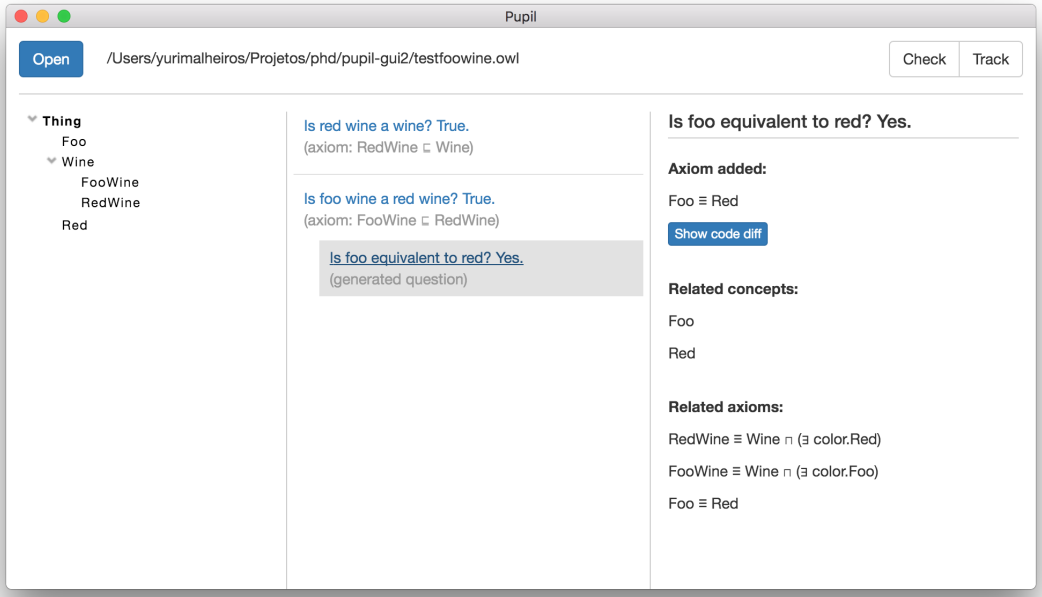

Figura 5.13: Tela da ferramenta mostrando o resultado da operação *coverage* para a QS "is foo equivalent to red?"

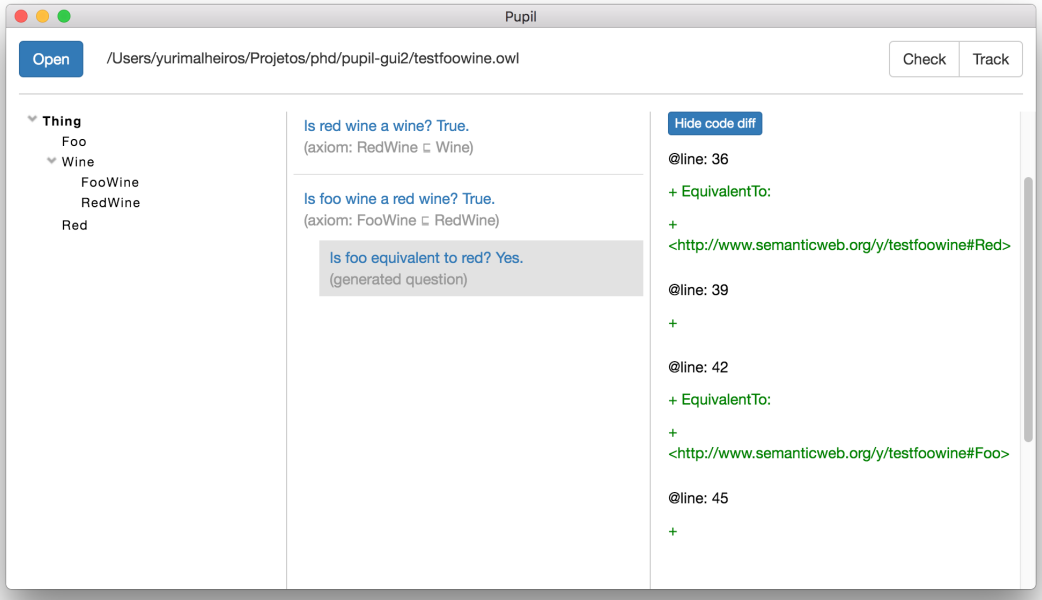

Figura 5.14: Tela da ferramenta mostrando o resultado da operação *diff* para a QS "is foo equivalent to red?"

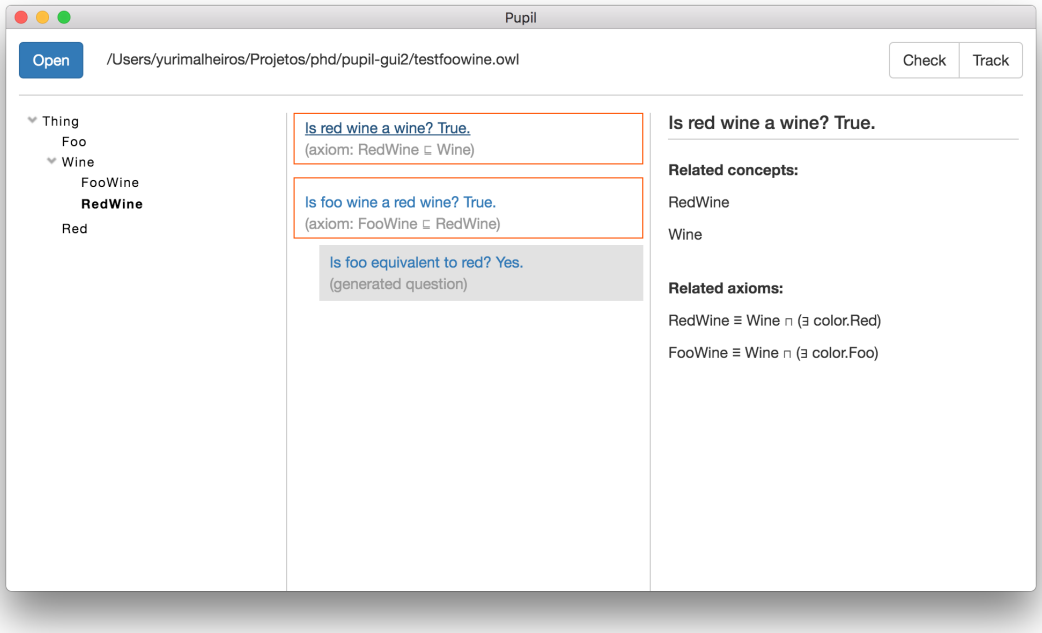

Figura 5.15: Tela da ferramenta mostrando o resultado da operação *Class-question* para a classe RedWine

6

Experimentos

Neste capítulo são apresentados os experimentos realizados para avaliar o método de expansão iterativa e interativa de ontologias e a sua implementação, apresentados nos capítulos 4 e 5 respectivamente. O Verificador e o Questionador foram testados de forma isolada para analisar suas capacidades de checar QCs e de gerar perguntas para adicionar conhecimento a uma ontologia. Além disso, o sistema desenvolvido foi testado através de dois estudos de caso.

6.1 Objetivo

O objetivo dos experimentos é avaliar o método de expansão iterativa de ontologias e a sua implementação proposta nesse trabalho. Para isso, foram definidas as seguintes questões para guiar os experimentos:

- **Q1:** O componente Verificador consegue checar QCs satisfeitas por axiomas presentes em uma ontologia?
- **Q2:** O componente Verificador consegue checar QCs satisfeitas por axiomas inferidos em uma ontologia?
- **Q3:** O componente Questionador consegue sugerir axiomas que ao serem adicionados a uma ontologia fazem uma QC representada por axioma antes não dedutível ser deduzido?
- **Q4:** O componente Questionador sugere axiomas similares aos axiomas criados por engenheiros?
- **Q5:** O sistema tem a capacidade de melhorar o processo de desenvolvimento de ontologias?

6.2 Artefatos

Nos experimentos foi utilizada a ontologia *Systematized Nomenclature of Medicine Clinical Terms* (SNOMED CT) mantida pela The International Health Terminology Standards Development Organisation (IHTSDO), uma ontologia em DL $\mathcal{EL}$. A única exceção foi o primeiro estudo de caso, no qual foi utilizada uma ontologia sobre profissões criada especialmente para os testes. A SNOMED CT é uma terminologia clínica que tem o potencial para melhorar a qualidade dos dados, prover informações para construção de sistemas de apoio a decisão e facilitar a interoperabilidade semântica através da captura de dados clínicos em um forma padronizada, inequívoca e granular (Lee *et al.* , 2014).

A SNOMED CT é a maior ontologia sobre conceitos clínicos existente; ela possui mais de 350.000 conceitos e 1,38 milhões de relações. O seu conteúdo é organizado em hierarquias (Sedano *et al.* , 2009), por exemplo, Clinical finding, Procedure, Body structure, Organism, Substance, etc.

Os componentes básicos da SNOMED CT são os conceitos, descrições e relações. Os conceitos são formulações sobre conhecimentos clínicos, todos eles representados por um identificador numérico único. Adicionalmente, os conceitos também possuem descrições em texto para que os seus significados sejam entendidos por pessoas. Um conceito pode ter várias descrições, uma principal e outras secundárias que são sinônimos da descrição principal. Por exemplo, o conceito Myocarditis, também possui a descrição Myocardial Inflammation, ou seja, Myocardial Inflammation é sinônimo de Myocarditis, de acordo com a SNOMED CT.

As relações são utilizadas para definir conceitos usando outros conceitos. Tais definições aumentam o valor dos dados representados na ontologia, permitindo que o seu conteúdo seja recuperado, pesquisado, reusado e analisado de diversas formas. As relações na SNOMED CT possuem o formato objeto-atributo-valor, representando que o objeto se relaciona com o valor de acordo com um atributo. A ontologia define diversas relações, entre elas temos a relação *is-a*, que é usada para definição de subclasses, por exemplo, *Calcium is-a Chemical Element*. Outra relação é a *finding-site*, que é utilizada para especificar em que parte do corpo um conceito possui efeito, por exemplo, *Appendicitis finding-site Appendix Structure*.

6.3 Preparação dos Artefatos

Antes de iniciar os testes do sistema utilizando a SNOMED CT, modificações foram realizadas na ontologia. Primeiramente, a SNOMED CT possui um formato próprio no qual suas definições estão representadas em um conjunto de arquivos texto. Como o sistema implementado nesse trabalho suporta ontologias em OWL, a SNOMED CT foi convertida para esse formato utilizando um script de conversão provido pela IHTSDO.

Os conceitos na SNOMED CT são representados por IDs numéricos com descrições adicionais em texto, porém, o sistema depende dos nomes dos conceitos para as QCs, QSs e

para apresentar os resultados das operações de rastreamento. Então, ter conceitos representados apenas por números dificulta o uso do sistema e torna seus resultados menos valiosos, assim, os conceitos representados por IDs numéricos na SNOMED CT foram substituídos por nomes de acordo com sua descrição. Para isso foram utilizadas duas regras. Para classes, todas as palavras da descrição principal tiveram suas primeiras letras convertidas para letras maiúsculas e em seguida foram removidos todos os espaços. Por exemplo, uma classe com descrição "Human body", teve seu nome convertido para "HumanBody". Para papéis, todas as palavras também tiveram suas primeiras letras convertidas para maiúscula, com exceção da primeira palavra, e em seguida foram removidos todos os espaços. Por exemplo, um papel com descrição "finding site", teve seu nome convertido para "findingSite". Ao final desse processo, todos os conceitos da SNOMED CT passaram a ter nomes legíveis para seres humanos ao invés de números.

Vários conceitos da SNOMED CT possuem restrições existenciais com papéis que aparecem mais de uma vez em uma mesma definição. Isso é comum nas definições de doenças, que podem localizar-se em diferentes locais do corpo (*findingSite*) com diferentes morfologias para cada local (*associatedMorphology*) (Spackman *et al.* , 2002). Por exemplo, o conceito AcquiredPhimosis, possui os seguintes possíveis pares *findingSite*, *associatedMorphology*:

- *findingSite* = PreputialStructure; *associatedMorphology* = Adhesion
- *findingSite* = PreputialStructure; *associatedMorphology* = Stenosis

Para especificar que um *findingSite* está associado a uma *associatedMorphology*, a SNOMED utiliza um papel especial chamado roleGroup que agrupa as restrições. Assim, ao invés de especificar:

$$\exists findingSite.PreputialStructure \sqcap \exists associatedMorphology.Adhesion \sqcap$$
$$\exists findingSite.PreputialStructure \sqcap \exists associatedMorphology.Stenosis$$

a SNOMED CT usa:

$$\exists roleGroup.(\exists findingSite.PreputialStructure \sqcap \exists associatedMorphology.Adhesion) \sqcap$$
$$\exists roleGroup.(\exists findingSite.PreputialStructure \sqcap \exists associatedMorphology.Stenosis)$$

Os *roleGroups* conseguem agrupar e retiram a ambiguidade das definições que apresentam associações entre restrições existenciais. Entretanto, a semântica da definição do conceito é prejudicada pela inserção de uma restrição existencial que tem como função apenas agrupar outras restrições. Por isso, nos testes realizados, as restrições com o papel *roleGroup* foram removidas, mas sem retirar o valor associado a restrição. Por exemplo, um conceito $X \equiv \exists roleGroup.(\exists r.Y \sqcap s.Z)$, foi transformado em: $X \equiv \exists r.Y \sqcap s.Z$.

6.4 Metodologia de Avaliação

Nessa seção é apresentada a metodologia de avaliação para cada uma das questões definidas na Seção 6.1. A SNOMED CT após as modificações detalhadas na Seção 6.3 foi utilizada em todos os testes, com exceção do primeiro estudo de caso. Como a SNOMED CT é uma ontologia muito extensa, com muitos conceitos e axiomas, o processamento utilizando um artefato do seu tamanho costuma ser lento e consome muita memória. Tal limitação ocorre por dois motivos principais, o primeiro pelo tamanho do arquivo que precisa ser carregado e manipulado pela OWL API e o segundo por que a unificação em DL $\mathcal{EL}$ é um problema NP. Por isso, nos testes foram usados módulos da SNOMED, ou seja, apenas um subconjunto dos seus axiomas. Os módulos foram criados usando a ferramenta OWL-ME (Del Vescovo *et al.* , 2010) que gera módulos de acordo com um conjunto de conceitos usando a técnica de extração de módulos baseada em localidade (Cuenca Grau *et al.* , 2008). Nas próximas subseções são explicados detalhes de cada teste, incluindo como os módulos da SNOMED foram criados, são apresentados os resultados obtidos e, para finalizar, é realizada a discussão dos resultados.

6.4.1 O componente Verificador consegue checar QCs satisfeitas por axiomas presentes em uma ontologia?

O primeiro teste realizado teve como objetivo avaliar a capacidade do componente Verificador em checar os axiomas de uma ontologia. Esse teste foi realizado utilizando três módulos distintos da SNOMED CT. Para gerar os módulos, foram escolhidos aleatoriamente três conjuntos de 10 classes presentes na ontologia. As classes escolhidas são apresentadas na Tabela 6.1. Em seguida, cada um dos conjuntos foi utilizado como entrada da ferramenta de extração de módulos, assim gerando três módulos. O primeiro módulo contendo 2023 axiomas e 209 classes, o segundo com 2628 axiomas e 525 classes e o terceiro com 1649 axiomas e 330 classes.

O próximo passo foi, para cada módulo, escolher randomicamente 20 axiomas. O Apêndice A traz todos os 60 axiomas escolhidos, do A.1 ao A.20 tem-se os axiomas do primeiro módulo, do A.21 ou A.40 tem-se os axiomas do segundo módulo e do A.41 ao A.60 tem-se os do terceiro módulo. Entre eles, tem-se axiomas simples definindo relação de subclasse entre duas classes, como:

$$AllergenClass \sqsubseteq Substance$$

também tem-se axiomas com restrições existenciais:

Tabela 6.1: Classes escolhidas para geração de módulos para os testes de Q1

Conjunto 1	Conjunto 2	Conjunto 3
SynovialStructure OfAnkleAndorFoot	AccidentCaused DirectlyByCandle	IrenaPuella
PlasmaTaurineMeasurement	ConstructDental SpecialTray	SalmonellaBudapest
DietAverage	EntireSuperiorArticularProcess OfFourthLumbarVertebra	SlowTransitConstipation
AspartateMeasurement	AccidentallyStungByJellyFish	XIntentionalSelfPoisoningBy AndExposureToHallucinogens OccurrenceAtOtherSpecifiedPlace
Adrenocorticotrophic HormonePrecursor	CrystalArthropathyOfKnee	VOtherGastrointestinal TractPathogenCarrier
Progressive PseudorheumatoidDysplasia	LichenificationOfVulva	LooseJointBodyInMultipleJoints
MechanicalOpenPleurodesis	StructureOfPeritubular CapillaryPlexus	RemovalOfTemporary CardiacPacemakerLead UsingFluoroscopicGuidance
MFibroadenomaNOS	PostauricularLymphNode	TissueSpecimenFromTestis
MouseUrineRASTTest	PolyodonSpatula	AustralianThickknee
MassOfPancreas	TraumaticPubic SymphysisSeparation	ShiftingAbdominalDullness

$$ProcedureOnSkeletalSystem \equiv ProcedureOnMusculoskeletalSystem \sqcap (\exists procedureSite.SkeletalSystemStructure)$$

além de axiomas com múltiplas conjunções:

$$\begin{aligned} CongenitalAnomalyOfBodyCavity \equiv\ & CongenitalAnomalyOfTrunk \sqcap Disease \\ & \sqcap ((\exists associatedMorphology.CongenitalAnomaly) \\ & \sqcap (\exists associatedMorphology.DevelopmentalAbnormality) \\ & \sqcap (\exists findingSite.BodyCavityStructure) \\ & \sqcap (\exists occurrence.Congenital)) \\ & \sqcap (\exists occurrence.Congenital) \end{aligned}$$

6.4.1.1 Resultados

Com os axiomas definidos, geraram-se manualmente as QCs necessárias para checar o conhecimento de cada axioma, ou seja, QCs que representam cada axioma. É importante observar que não existem QCs no sistema que chequem perguntas com operadores de conjunção, por isso nos axiomas que possuem tal operador foi necessário gerar mais de uma pergunta. Por

exemplo, um axioma $A \sqsubseteq B \sqcap C$ precisa das perguntas “is A a B?” e “is A a C?” para ser checado completamente. A Tabela 6.2 traz as perguntas necessárias para checar os primeiros 20 axiomas escolhidos, a Tabela A.1 do Apêndice traz a lista completa com as QCs para todos os 60 axiomas. A primeira coluna da Tabela 6.2 mostra o número do axioma de acordo com sua referência no Apêndice A, a segunda coluna apresenta as questões de competência e a terceira as respostas esperadas.

Tabela 6.2: Questões de competência geradas para checar os 20 primeiros axiomas escolhidos aleatoriamente no teste do Verificador

Axioma	Questão de Competência	Resposta Esperada
A.1	Is allergen class a substance?	True
A.2	Is drug or medicament a substance?	True
A.3	Is amino acid supplement a protein supplementation?	True
A.4	Is synovial structure of limb a limb structure?	True
	Is synovial structure of limb a regional synovial structure?	True
A.5	Is congenital anomaly of body cavity a congenital anomaly of trunk?	True
	Is congenital anomaly of body cavity a disease?	True
	What is the associated morphology of congenital anomaly of body cavity?	Congenital anomaly
	Does congenital anomaly of body cavity have associated morphology some developmental abnormality?	True
	Where is the finding site of congenital anomaly of body cavity?	Body cavity structure
	What is the occurrence of congenital anomaly of body cavity?	Congenital
A.6	Is dysplasia a growth alteration?	True
	Is dysplasia a lesion?	True
A.7	Is procedure on skeletal system a procedure on musculoskeletal system?	True
	What is the procedure site of procedure on skeletal system?	skeletal system structure
A.8	Is abdomen destructive procedure a destructive procedure?	True
	Is abdomen destructive procedure a procedure on abdomen?	True

	What is the method of abdomen destructive procedure?	Destruction action
	Where is the procedure site direct of abdomen destructive procedure?	Abdominal structure
A.9	Is serous sac structure a body organ structure?	True
A.10	Is leg destructive procedure a destructive procedure?	True
	What is the method of leg destructive procedure?	Destruction action
	Where is the procedure site direct of leg destructive procedure?	Lower leg structure
A.11	Is finding of region of thorax a finding of trunk structure?	True
	What is the finding site of finding of region of thorax?	Thoracic structure
A.12	Is drug allergen or pseudoallergen an allergen or pseudoallergen?	True
A.13	Is abdominal cavity structure an abdominal structure?	True
	Is abdominal cavity structure a body cavity structure?	True
	Is abdominal cavity structure a structure of body cavity subdivision?	True
	Is abdominal cavity structure a structure of compartment of abdomen?	True
A.14	Is disorder of bone a disorder of skeletal system?	True
	What is the finding site of disorder of bone?	Bone structure
A.15	Is lower trunk structure a lower body part structure?	True
	Is lower trunk structure a structure of subregion of trunk?	True
A.16	Is right lower extremity structure a lower limb structure?	True
	Is right lower extremity structure a structure of right half of body?	True
A.17	Is structure of viscus a body organ structure?	True

A.18	Is destruction action a surgical action?	True
A.19	Is surgical procedure on thorax a procedure on thorax?	True
	Is surgical procedure on thorax a surgical procedure?	True
	Does surgical procedure on thorax have method some surgical action?	True
	Where is the procedure site of surgical procedure on thorax?	Thoracic structure
A.20	Is fixation a procedure by method?	True
	What is the method of fixation?	Fixation action

6.4.1.2 Discussão

O sistema desenvolvido foi capaz de checar todos os axiomas selecionados aleatoriamente. Nos resultados apresentados foram usadas as QCs dos tipos: "Is a", "Have property" e "Wh question" (*What* e *Where*) As QCs usadas dependem da nomenclatura utilizada na ontologia, na SNOMED CT, os três tipos de perguntas foram suficientes para checar o conhecimento representado pelos 60 axiomas. Nem sempre isso será possível, pois ontologias com nomes de classes e papéis diferentes podem requerer questões diferentes. Os tipos adicionais de questões também trazem uma flexibilidade maior para o usuário, podendo ele escolher o que mais se adéqua ao conhecimento que ele quer checar.

As QCs do sistema não tratam conjunções, mas essa limitação é contornada usando mais de uma QC por axioma, uma para cada componente da conjunção. Por exemplo, o axioma a seguir possui três operadores de conjunção:

$$\begin{aligned} AbdomenDestructiveProcedure \equiv\ & DestructiveProcedure \\ & \sqcap ProcedureOnAbdomen \\ & \sqcap (\exists method.DestructionAction) \\ & \sqcap (\exists procedureSiteDirect.AbdominalStructure) \end{aligned}$$

Não é possível gerar uma QC que cheque todo o axioma de uma vez, entretanto pode-se formular quatro QCs para checar o axioma. A relação entre *AbdomenDestructiveProcedure* e *DestructiveProcedure* pode ser checada pela QC "Is abdomen destructive procedure a destructive procedure?" com a resposta "true"; A relação entre *AbdomenDestructiveProcedure* e *ProcedureOnAbdomen* pode ser checada pela QC "Is abdomen destructive procedure a proce-

dure on abdomen?" com a resposta "true"; a relação entre *AbdomenDestructiveProcedure* e $\exists method.DestructionAction$ pode ser checada pela QC "What is the method of abdomen destructive procedure?" com a resposta "Destruction action" e a relação entre *AbdomenDestructiveProcedure* e $\exists procedureSiteDirect.AbdominalStructure$ pode ser checada pela QC "Where is the procedure site direct of abdomen destructive procedure?" com a resposta "Abdominal structure".

Com padrões simples de nomenclatura de classes e papéis é possível usar as questões de competência do sistema para checar o conhecimento expressado pelos axiomas da ontologia. Dessa forma, um engenheiro pode ter QCs que cobrem todo o conhecimento de uma ontologia em DL $\mathcal{EL}$. Nos testes, vimos que os axiomas que especificam relação de subclasse ou equivalência podem ser testados usando a pergunta "Is a" e os axiomas com restrições existenciais podem ser testados usando a QCs dos tipos "Have property" e "Wh question".

Pode-se resumir o uso das QCs do sistema da seguinte forma. Dado um axioma $X \sqsubseteq A_1 \sqcap A_2 \sqcap ... \sqcap A_n \sqcap \exists r.B_1 \sqcap \exists r.B_2 \sqcap ... \sqcap \exists B_n$, as relações $X \sqsubseteq A_i$ podem ser checadas com as perguntas do tipo: "Is a", "Are" e "Which exist", já as relações $X \sqsubseteq \exists r.B_i$ podem ser checadas com as perguntas do tipo: "Have property", "Verb property", "Preposition property", "Wh question".

6.4.2 O componente Verificador consegue checar QCs satisfeitas por axiomas inferidos em uma ontologia?

Além de checar QCs que são respondidas através de axiomas codificados na ontologia, o Verificador precisa ser capaz de checar QCs que são respondidas através de axiomas que podem ser inferidos; para isso o componente utiliza um raciocinador. Nesse teste, foram usados os mesmos três módulos da SNOMED CT do teste anterior.

Em seguida, para cada módulo, foi utilizada a OWL API para obtenção de todas as novas relações inferidas pelo raciocinador ELK. Para o primeiro módulo, foram inferidas 757 relações, para o segundo módulo 994 e para o terceiro 599. Com esse conjunto de axiomas inferidos em mãos foram selecionados randomicamente 10 axiomas para cada um dos módulos. O Apêndice A traz todos os 30 axiomas (10 para cada um dos três módulos). Do axioma A.61 até o A.70 tem-se os axiomas do primeiro módulo, do A.71 ao A.80 tem-se os axiomas do segundo módulo e do A.81 ao A.90 tem-se os do terceiro módulo. Os axiomas inferidos são relações simples de subclasse ou equivalência, por exemplo, entre os axiomas escolhidos randomicamente, temos: $BloodMaterial \sqsubseteq HematologicAgent$ e $StructureOfLeftTestis \equiv StructureOfRightTestis$.

6.4.2.1 Resultados

Com os axiomas definidos, geraram-se manualmente as QCs necessárias para checar o conhecimento de cada axioma. A Tabela 6.3 traz as perguntas necessárias para checar cada um dos 30 axiomas inferidos escolhidos aleatoriamente. A primeira coluna da Tabela 6.3 mostra o

número do axioma de acordo com sua referência no Apêndice A, a segunda coluna apresenta as questões de competência e a terceira as respostas esperadas.

Tabela 6.3: Questões de competência geradas para checar os axiomas inferidos escolhidos randomicamente

Axioma	Questão de Competência	Resposta Esperada
A.61	Is blood material a hematologic agent?	True
A.62	Is mass of body structure a finding by site?	True
A.63	Is surgical procedure on lower extremity a limb operation?	True
A.64	Is amino acid supplement a protein supplementation?	True
A.65	Is procedure on soft tissue a procedure by site?	True
A.66	Is bone structure a bone andor joint structure?	True
A.67	Is structure of subregion of trunk a structure of left side of trunk	True
A.68	Is mass of soft tissue a general finding of soft tissue	True
A.69	Is pleura finding a finding of region of thorax	True
A.70	Is destructive procedure on lower limb a destructive procedure by anatomic site	True
A.71	Is knee joint structure a structure of left knee?	True
A.72	Is structure of right side of vulva a lower female genital structure?	True
A.73	Is disorder of skeletal system a disease of musculoskeletal system?	True
A.74	Is skin and subcutaneous tissue structure a structure of right half of body?	True
A.75	Is lichen a papular eruption?	True
A.76	Is kidney structure a kidney and or ureter structures?	True
A.77	Is right kidney structure a structure of viscus of abdomen proper?	True
A.78	Is disorder of abdominal wall a disorder of body wall?	True
A.79	Is disease of lymph node a disorder of lymphoid system?	True

A.80	Is bone structure a skeletal system structure?	True
A.81	Is cardiovascular structure of trunk a trunk structure?	True
A.82	Is procedure by device a procedure?	True
A.83	Is structure of body cavity subdivision a body cavity structure?	True
A.84	Is fluoroscopy of systems a procedure on body system?	True
A.85	Is structure of right side of trunk a body part structure?	True
A.86	Is structure of left half of abdomen a structure of abdominal segment of trunk?	True
A.87	Is structure of left testis a structure of right testis?	True
	Is structure of left testis a testis structure?	True
	is structure of left testis a unilateral testis?	True
A.88	Is disorder of pelvic region of trunk a disorder of trunk?	True
A.89	Is specimen from testis a gonad sample?	True
A.90	Is disease of mediastinum a mediastinal finding?	True

6.4.2.2 Discussão

Os axiomas inferidos pelo raciocinador são axiomas simples de subsunção e de equivalência. Nenhum desses axiomas estão especificados na ontologia, mas todos eles podem ser inferidos. Então, conseguir checar esse tipo de axioma mostra que o Verificador vai além de simplesmente procurar axiomas já codificados, ele também usa um raciocinador nas suas operações de checagem. Novamente todos os axiomas foram checados corretamente, ou seja, o Verificador conseguiu cobrir o conhecimento representado pelos 30 axiomas inferidos selecionados aleatoriamente.

Este tipo de checagem é importante, pois podem existir QCs que já são respondidas sem que um engenheiro tenha codificado um axioma explicitamente que represente a pergunta. Logo, engenheiros poderiam descobrir que o conhecimento já está representado através de uma checagem usando um raciocinador. Além disso, poder checar a validade dessas QCs pode evitar redundância e retrabalho na representação.

6.4.3 O componente Questionador consegue sugerir axiomas que ao serem adicionados a uma ontologia fazem uma QC representada por axioma antes não dedutível ser deduzido?

Para testar a capacidade do componente Questionador em gerar perguntas que correspondem a axiomas que ao serem adicionados a uma ontologia fazem uma QC não satisfeita ter seu axioma correspondente deduzido, foram efetuados os passos descritos a seguir.

Primeiramente foram escolhidos aleatoriamente três conjuntos de 20 classes presentes na ontologia. As classes escolhidas são apresentadas na Tabela 6.4. Em seguida, cada um dos conjuntos foi utilizado como entrada da ferramenta de extração de módulos, dessa forma gerando três módulos distintos. O primeiro módulo contendo 3832 axiomas e 775 classes, o segundo com 4492 axiomas e 913 classes e o terceiro com 4738 axiomas e 958 classes.

O próximo passo foi, para cada módulo, escolher aleatoriamente 10 axiomas de equivalência. O Apêndice A traz todos os 30 axiomas escolhidos, do A.91 ao A.100 tem-se os axiomas do primeiro módulo, do A.101 ou A.110 tem-se os axiomas do segundo módulo e do A.111 ao A.120 tem-se os do terceiro módulo. Os axiomas escolhidos possuem definições de equivalência e usam operadores de conjunção e restrição existencial. Por exemplo, entre os axiomas escolhidos temos:

$$DiseaseOfMusculoskeletalSystem \equiv DisorderByBodySite \sqcap (\exists findingSite.StructureOfMusculoskeletalSystem)$$

e

$$MalignantNeoplasmOfGenitourinaryOrgan \equiv DiseaseOfTheGenitourinarySystem \sqcap ((\exists associatedMorphology.MalignantNeoplasmMorphology) \sqcap (\exists findingSite.StructureOfGenitourinarySystem))$$

Para cada um dos axiomas escolhidos foi gerado um novo módulo retirando o axioma escolhido do módulo original a que ele pertencia, dessa forma foram gerados 30 novos módulos. Assim, com a remoção dos axiomas, foram injetadas falhas na ontologia para testar o componente Questionador.

Para ter certeza que o conhecimento representado pelos axiomas selecionados foi realmente removido da ontologia, foram feitos testes para verificar se o axioma removido ainda podia ser inferido pelos axiomas restantes. Em todos os casos não foi possível inferir o axioma removido.

Continuando, para testar o Questionador é necessário formular QCs. Para esse experi-

Tabela 6.4: Classes escolhidas para geração de módulos para os testes de Q3

Conjunto 1	Conjunto 2	Conjunto 3
AbilityToSitDown	ProstateEnlargedOnPR	CongenitalDeformity OfForehead
AcuteRejectionOf LungTransplant	CellulitisAndAbscess OfPerineum	RevisionOfAnastomosisO fStomachToJejunumNEC
ReactiveArthropathy OfWrist	QuercusVelutina	NasalPackingHemorrcontrol
GenerallyContracted PelvisUnspecified	ApplicationOfPlaster FigureOfEightShortArm	NasalCatheter
Signalaveraged Electrocardiography	Chaetomium Hexagonosporum	EntireIntermetatarsalJoint
FGSyndrome	MeasurementOfBovine LeukemiaVirusAntibody	MycoplasmaBuccale
StructureOfLigament OfAuricle	ApplicationOfCrutchfield TongsOfSkullWith SynchronousSkeletalTraction	LongFaceSyndrome
TabletForOralSuspension	QRSAxisFinding	SuperficialBurnOf WristAndHand
FollicularCenterBcell LymphomaNodalsystemic WithSkinInvolvement	UretericStoneSample	DermatosisDueToTick
OnExaminationExternal FemaleGenitaliaNo AbnormalityDetected	PrematureBirthOf NewbornQuadruplets	StructureOfShoulder GirdleRegion
HighPotassiumFood	Intracervical ArtificialInsemination	ReimplantationOf MitralPapillaryMuscle
MethionylAminopeptidase	OpenReduction OfMaxillaryFracture	MultipleComplications DueToDiabetesMellitus
CampylobacterConcisus	TotalOphthalmoplegia	HemisphericCerebellar Agenesis
UGymin	UretericFistulaToCervix	BipolarisHawaiiense
UnableToTurnOn DomesticAppliance	HeparinSodium UnitsmLPatencySolution	Epididymovasostomy
SuboccipitalApproach	ImpairedIntestinal FatAbsorption	WholeChromosome TrisomyMitotic NondisjunctionMosaicism
OctreotideAllergy	ExcisionOfCranialTumor	PlasmaViscosity BorderlineRaised
Smithi	SampleVirusIdentified	ClassicalSwine FeverVirusRNA
UreaseTestWayneMethod	LigationAndStripping OfVaricoseVeinOfThorax	Omicronpapillomavirus
IntentionalTobramycin Poisoning	JotkoCattle	BiopsyAdmin

mento foi usada a seguinte abordagem para formular as questões. Cada axioma escolhido define que uma classe X é equivalente a alguma expressão. Dado isto, para formular as QCs, foram recuperadas todas as superclasses diretas de X. Assim, foram geradas QCs que questionam se X é subclasse das superclasses diretas. Com a remoção do axioma essa relação não é mais verdadeira como no módulo sem a remoção. Adicionalmente, as superclasses diretas podem possuir definições de equivalência da forma: $Y \equiv A_1 \sqcap A_2 \sqcap ... \sqcap A_n$. Nesse caso, também foram geradas QCs que questionam se X é subclasse de $A_1, A_2, ..., A_n$.

Através desse processo foram definidas uma ou mais QCs para cada um dos 30 novos módulos gerados após a remoção dos axiomas. Por exemplo, para o axioma que define DiseaseOfMusculoskeletalSystem mostrado anteriormente, foram geradas as seguintes QCs com suas respectivas respostas:

- Is disease of musculoskeletal system a disorder of body system? True.
- Is disease o musculoskeletal system a disease? True.
- What is the finding site of disease of musculoskeletal system? Body system structure.
- Is disease of musculoskeletal system a musculoskeletal finding? True.
- Is disease of musculoskeletal system a finding by site? True.
- What is the finding site of disease of musculoskeletal system? Structure of musculoskeletal system.

Já para o axioma que define MalignantNeoplasmOfGenitourinaryOrgan, as QCs geradas com suas respectivas respostas foram:

- Is malignant neoplasm of genitourinary organ a disease of the genitourinary system? True.
- Is malignant neoplasm of genitourinary organ a disorder of trunk? True.
- What is the finding site of malignant neoplasm of genitourinary organ? Structure of genitourinary system.
- Is malignant neoplasm of genitourinary organ a malignant tumor of pelvis? True.
- Is malignant neoplasm of genitourinary organ a malignant neoplastic disease? True.
- What is the associated morphology of malignant neoplasm of genitourinary organ? Malignant neoplasm morphology.
- What is the finding site of malignant neoplasm of genitourinary organ? Pelvic structure.

- Is malignant neoplasm of genitourinary organ a mass of urogenital structure? True.
- Is malignant neoplasm of genitourinary organ a mass of body structure? True.
- Is malignant neoplasm of genitourinary organ a urogenital finding? True.
- What is the associated morphology of malignant neoplasm of genitourinary organ? Mass.
- What is the finding site of malignant neoplasm of genitourinary organ? Structure of genitourinary system

6.4.3.1 Resultados

Com as QCs escolhidas para cada módulo em mãos, o sistema foi utilizado para checá-las. Cada QC foi checada na sua ontologia correspondente (módulo com axioma removido). Como nenhuma delas foi satisfeita, dado que os axiomas que representavam o conhecimento questionado pelas perguntas foram retirados, então o sistema ativou o componente Questionador para gerar as QSs. Para cada QC utilizada o sistema gerou uma pergunta quando um unificador era encontrado ou nenhuma quando não foi possível encontrar um unificador. A Tabela 6.5 mostra as QSs geradas, e os seus axiomas correspondentes, para as QCs criadas para os axiomas que definem DiseaseOfMusculoskeletalSystem e MalignantNeoplasmOfGenitourinaryOrgan mostrados anteriormente. É importante ressaltar, que nem toda QC gera uma QS, pois existem casos que a unificação não retorna unificador.

Tabela 6.5: Questões do Sistema geradas e seus axiomas correspondentes para o primeiro módulo

Questões do Sistema	Axioma da Questão do Sistema
Does a disease of musculoskeletal system have finding site some body system structure?	$DiseaseOfMusculoskeletalSystem \equiv \exists findingSite.BodySystemStructure$
Is a disease of musculoskeletal system equivalent to a clinical finding, that have finding site some anatomical or acquired body structure, that have finding site some anatomical structure, that have finding site some structure of musculoskeletal system?	$DiseaseOfMusculoskeletalSystem \equiv ClinicalFinding \sqcap (\exists findingSite.AnatomicalOrAcquiredBodyStructure) \sqcap (\exists findingSite.AnatomicalStructure) \sqcap (\exists findingSite.StructureOfMusculoskeletalSystem)$
Is a disease of musculoskeletal system equivalent to a clinical finding, that have finding site some anatomical or acquired body structure, that have finding site some anatomical structure?	$DiseaseOfMusculoskeletalSystem \equiv ClinicalFinding \sqcap (\exists findingSite.AnatomicalOrAcquiredBodyStructure) \sqcap (\exists findingSite.AnatomicalStructure)$
Does a disease of musculoskeletal system have finding site some structure of musculoskeletal system?	$DiseaseOfMusculoskeletalSystem \equiv \exists findingSite.StructureOfMusculoskeletalSystem$

Does a malignant neoplasm of genitourinary organ have finding site some structure of genitourinary system?	$MalignantNeoplasmOfGenitourinaryOrgan \equiv \exists findingSite.StructureOfGenitourinarySystem$
Does a malignant neoplasm of genitourinary organ have finding site some pelvic structure?	$MalignantNeoplasmOfGenitourinaryOrgan \equiv \exists findingSite.PelvicStructure$
Is a malignant neoplasm of genitourinary organ equivalent to a clinical finding, that have associated morphology some mass, that have finding site some anatomical or acquired body structure, that have finding site some anatomical structure, that have finding site some structure of genitourinary system?	$MalignantNeoplasmOfGenitourinaryOrgan \equiv ClinicalFinding \sqcap (\exists associatedMorphology.Mass) \sqcap (\exists findingSite.AnatomicalOrAcquiredBodyStructure) \sqcap (\exists findingSite.AnatomicalStructure) \sqcap (\exists findingSite.StructureOfGenitourinarySystem)$
Is a malignant neoplasm of genitourinary organ equivalent to a clinical finding, that have associated morphology some mass, that have finding site some anatomical or acquired body structure, that have finding site some anatomical structure?	$MalignantNeoplasmOfGenitourinaryOrgan \equiv ClinicalFinding \sqcap (\exists associatedMorphology.Mass) \sqcap (\exists findingSite.AnatomicalOrAcquiredBodyStructure) \sqcap (\exists findingSite.AnatomicalStructure)$
Is a malignant neoplasm of genitourinary organ equivalent to a clinical finding, that have finding site some anatomical or acquired body structure, that have finding site some anatomical structure, that have finding site some structure of genitourinary system?	$MalignantNeoplasmOfGenitourinaryOrgan \equiv ClinicalFinding \sqcap (\exists findingSite.AnatomicalOrAcquiredBodyStructure) \sqcap (\exists findingSite.AnatomicalStructure) \sqcap (\exists findingSite.StructureOfGenitourinarySystem)$
Does a malignant neoplasm of genitourinary organ have associated morphology some mass?	$MalignantNeoplasmOfGenitourinaryOrgan \equiv \exists associatedMorphology.Mass$
Does a malignant neoplasm of genitourinary organ have finding site some structure of genitourinary system?	$MalignantNeoplasmOfGenitourinaryOrgan \equiv \exists findingSite.StructureOfGenitourinarySystem$

Para o primeiro módulo foram perguntadas 63 QCs. 38 delas geraram QSs que, ao serem respondidas positivamente, adicionaram conhecimento para satisfazer a QC perguntada, e para as 25 QCs restantes não foram encontrados unificadores para os axiomas que elas correspondem, portanto nenhuma QS foi gerada pelo sistema. Para o segundo módulo foram perguntadas 54 QCs. 36 delas geraram QSs que, ao serem respondidas positivamente, adicionaram o conhecimento para satisfazer a QC e 18 não geraram unificadores. Por fim, para o terceiro módulo foram utilizadas 66 QCs, dentre elas 53 geraram QSs para adicionar o conhecimento necessário e 13 não geraram unificadores.

A Tabela 6.6 sumariza os resultados da geração de perguntas pelo Questionador. Sua

primeira coluna especifica o módulo em que os testes foram feitos, a segunda coluna traz a quantidade e a porcentagem de QCs que passaram a ser satisfeitas após a adição de conhecimento através de uma reposta a uma QS. A terceira coluna traz a quantidade e a porcentagem das QCs sem unificador e a última coluna apresenta o total de QCs.

Tabela 6.6: Sumarização dos resultados da geração de perguntas pelo Questionador

Módulo	Satisfeito	Sem unificador	Total
1	38 (60,32%)	25 (39,68%)	63
2	36 (66,67%)	18 (33,33%)	54
3	53 (80,30%)	13 (19,69%)	66

6.4.3.2 Discussão

Nos resultados obtidos, o Questionador consegue através da unificação sugerir desde perguntas simples relacionando duas classes através de um papel como: “does a disease of musculoskeletal system have finding site some body system structure?” ou “does a fistula of uterus have finding site some uterine structure?” até perguntas complexas com diversas conjunções, classes e restrições como: “is a malignant neoplasm of genitourinary organ equivalent to a clinical finding, that have associated morphology some mass, that have finding site some anatomical or acquired body structure, that have finding site some anatomical structure?”.

Perguntas menores podem ser entendidas mais facilmente pelo usuário, já as perguntas grandes e mais complexas podem ser um problema, já que existem muitas classes e papéis envolvidos e diversas relações entre eles representadas na pergunta. Isso acontece, pois, como todo o axioma de equivalência é representado pela pergunta, é necessário que a definição completa seja mostrada de uma só vez.

Mesmo que uma QS traga um axioma que adicionado à ontologia faça uma QC ser respondida corretamente, cabe ao engenheiro de ontologias julgar se o que a QS está especificando é correto no escopo da ontologia desenvolvida. A unificação garante que a QC vai passar a ser deduzida, mas ela não garante que os axiomas estejam ontologicamente corretos. Por isso, o Questionador faz uma sugestão ao engenheiro, ele sempre tem a opção de responder positivamente ou não uma QS.

6.4.4 O componente Questionador sugere axiomas similares aos axiomas criados por engenheiros?

Além de testar se o Questionador consegue gerar perguntas que ao serem respondidas adicionam conhecimento que vai satisfazer uma QC perguntada anteriormente, também foi avaliado se os axiomas sugeridos são semelhantes aos axiomas criados por engenheiros de ontologias.

Na subseção anterior foi mostrado todo o processo para escolher as QCs e gerar QSs. Para cada módulo foram geradas 10 novas ontologias, cada uma retirando um determinado axioma. Em seguida a QC foi perguntada de acordo com o axioma retirado e, por fim, as QSs foram geradas e seus axiomas correspondentes foram adicionados à ontologia para fazer a QC inicial ser satisfeita. Esses módulos, QCs e QSs foram utilizados também nos testes de similaridade.

Para avaliar se o Questionador gera axiomas similares aos dos engenheiros, para cada QS gerada foi comparado o seu axioma correspondente com o axioma que foi retirado de cada ontologia, já que ele foi codificado pelos engenheiros da SNOMED CT. Desse modo, tem-se uma forma de analisar as semelhanças e diferenças entre axiomas gerados automaticamente através da unificação e os axiomas criados manualmente.

Para comparar os axiomas foram utilizadas duas abordagens. Na primeira, os axiomas foram transformados em conjuntos de classes e papéis de acordo com as classes e papéis presentes neles. Por exemplo, dado um axioma $X \equiv A \sqsubseteq \exists r.B$, o conjunto gerado foi: $\{X, A, r, B\}$, ou seja, cada elemento do conjunto é uma classe ou um papel no axioma. Na segunda abordagem, um conjunto também foi criado, mas os papéis não foram dissociados das classes a que estão relacionados, por exemplo, dado o mesmo axioma anterior $X \equiv A \sqsubseteq \exists r.B$, o conjunto gerado foi: $\{X, A, r.B\}$. Note que agora tem-se o elemento $r.B$ e não dois elementos r e B. Com isso, para saber o grau de semelhança entre dois axiomas, comparou-se os conjuntos de conceitos de cada um utilizando as duas abordagens.

As métricas usadas para comparar os conjuntos foram a precisão, abrangência e f-measure. A precisão mede a proporção dos elementos que estão presentes no conjunto do axioma sugerido (S_s) que também estão no conjunto do axioma codificado pelos engenheiros (S_c). Por exemplo, se $S_s = \{A, B, X, Y\}$ e $S_c = \{A, B, C\}$, então a precisão é $2/4$, pois dos quatro conceitos de S_s, dois estão presentes em S_c. A abrangência mede a proporção dos elementos que estão presentes em S_c que estão em S_s. Usando o exemplo anterior, a abrangência é $2/3$, pois dos três elementos em S_c, dois estão presentes em S_s. A f-measure é a média harmônica da precisão (P) e a da abrangência (A), ela é calculada pela seguinte fórmula: $F = 2 \times \frac{P \cdot A}{P + A}$.

6.4.4.1 Resultados

A Tabela 6.7 mostra, para cada QS da Tabela 6.5, qual a semelhança do axioma gerado em relação ao axioma original utilizando as métricas de precisão, abrangência e f-measure. Cada métrica aparece duas vezes, as métricas seguidas de (1) foram calculadas usando a primeira abordagem de comparação, e as métricas seguidas de (2) foram calculadas usando a segunda abordagem. Cada linha da Tabela 6.7 traz os resultados para a QS que se encontra na linha correspondente na Tabela 6.5.

Para sumarizar os resultados foi calculada a média das precisões, abrangências e f-measures. Novamente, cada métrica aparece duas vezes, com (1) as métricas usando a primeira abordagem de comparação e com (2) usando a segunda. É importante ressaltar que os axiomas

Tabela 6.7: Métricas de semelhança entre os axiomas originais e os axiomas apresentados na Tabela 6.5

Precisão (1)	Abrangência (1)	F-measure (1)	Precisão (2)	Abrangência (2)	F-measure (2)
50,00%	33,33%	40,00%	0,00%	0,00%	0,00%
40,00%	66,67%	50,00%	25,00%	50,00%	33,33%
25,00%	33,33%	28,57%	0,00%	0,00%	0,00%
100,00%	66,67%	80,00%	100,00%	50,00%	66,67%
100,00%	40,00%	57,14%	100,00%	33,33%	50,00%
50,00%	20,00%	28,57%	0,00%	0,00%	0,00%
42,86%	60,00%	50,00%	20,00%	33,33%	25,00%
33,33%	40,00%	36,36%	0,00%	0,00%	0,00%
40,00%	40,00%	40,00%	25,00%	33,33%	28,57%
50,00%	20,00%	28,57%	0,00%	0,00%	0,00%
100,00%	40,00%	57,14%	100,00%	33,33%	50,00%

Tabela 6.8: Sumarização dos resultados de comparação dos axiomas gerados com os axiomas originais

Módulo	Precisão (1)	Abrangência (1)	F-measure (1)	Precisão (2)	Abrangência (2)	F-measure (2)
1	56,88%	30,31%	35,09%	35,26%	12,15%	15,81%
2	50,42%	35,09%	36,80%	27,41%	15,48%	16,50%
3	46,69%	33,33%	35,09%	21,86%	11,32%	12,59%

sem unificadores não foram levados em consideração nesse cálculo. A Tabela 6.8 apresenta os resultados sumarizados.

6.4.4.2 Discussão

Ao comparar os axiomas sugeridos pelo Questionador com os axiomas que estavam originalmente na SNOMED CT, pode-se perceber que o componente consegue sugerir alguns conceitos que estavam no axioma original, mas também sugere outros que não estavam ou deixa de sugerir conceitos que deveria. Nota-se pelos resultados que os valores de precisão são maiores que os de abrangência na média, o que evidencia que as sugestões, apesar de definirem conceitos corretos, deixam de lado conceitos importantes de acordo com a definição original.

Por exemplo, na ontologia tem-se o axioma:

$$
\begin{aligned}
&CongenitalFemaleUrogenitalAnomaly \equiv \\
&Disease \sqcap \\
&GenitourinaryCongenitalAnomalies \sqcap \\
&\exists associatedMorphology.CongenitalAnomaly \sqcap \\
&\exists associatedMorphology.DevelopmentalAbnormality \sqcap \\
&\exists findingSite.FemaleGenitourinarySystemStructure \sqcap \\
&\exists occurrence.Congenital
\end{aligned}
$$

e o Questionador sugere:

$$CongenitalFemaleUrogenitalAnomaly \equiv \exists associatedMorphology.DevelopmentalAbnormality$$

A precisão, nesse caso, é 100% nas duas abordagens, pois o que o foi sugerido está no axioma original, por outro lado, o Questionador deixou de sugerir vários conceitos presentes no axioma original, o que levou a abrangência ter valor 22.22% na primeira abordagem e 16.67% na segunda.

Na abordagem que separa todos conceitos, incluindo papéis das suas classes relacionadas, os valores de precisão e abrangência foram maiores. Isso mostra que em certos casos o Questionador sugere uma restrição existencial que contém um papel que estava no axioma original, mas não contém a classe associada a esse papel. Por exemplo, na ontologia tem-se o axioma:

$$\begin{aligned} &DiseaseOfMusculoskeletalSystem \equiv \\ &DisorderByBodySite \sqcap \\ &\exists findingSite.StructureOfMusculoskeletalSystem \end{aligned}$$

e o Questionador sugere:

$$DiseaseOfMusculoskeletalSystem \equiv \exists findingSite.BodySystemStructure$$

O papel $findingSite$ está presente nos dois, mas os seus valores são diferentes. O inverso também pode acontecer, o Questionador sugere um conceito, mas ele não possui um papel relacionado a ele numa restrição existencial da mesma maneira que no axioma original.

Também existem casos, que apesar do conceito sugerido e o original serem diferentes, eles possuem relação de subclasse ou superclasse. Por exemplo, a ontologia original possui o axioma:

$$\begin{aligned} &NeoplasmOfFace \equiv DisorderOfFace \sqcap \\ &NeoplasmOfHead \sqcap \\ &\exists associatedMorphology.Neoplasm \sqcap \\ &\exists findingSite.FaceStructure \end{aligned}$$

e o Questionador sugeriu:

$$NeoplasmOfFace \equiv \exists findingSite.HeadStructure$$

Note que o papel $findingSite$ em um axioma tem valor FaceStructure e no outro HeadStructure. Esses conceitos na ontologia possuem a relação: $FaceStructure \sqsubseteq HeadStructure$. Apesar de não ser o mesmo conceito, o axioma faz sentido, pois $FaceStructure$ é uma $HeadStructure$.

Com isso, pode-se concluir que o Questionador consegue sugerir algumas relações importantes para a SNOMED CT, entretanto ele tende a não oferecer uma pergunta que adicione todos os conceitos necessários para definir uma classe. Isso mostra mais uma vez o potencial do Questionador em ser um guia para o engenheiro, não uma palavra final sobre como definir uma classe. Por esse motivo, o Questionador permite que o usuário analise a pergunta antes de concordar se deseja adicionar o conhecimento sugerido a ontologia.

6.4.5 O sistema tem a capacidade de melhorar o processo de desenvolvimento de ontologias?

Para responder a última questão levantada no início do capítulo foram realizados dois estudos de caso exploratórios. O objetivo de ambos foi descobrir pontos fortes e fracos da abordagem desenvolvida e da ferramenta, com o foco na produtividade do desenvolvedor e na qualidade das ontologias desenvolvidas. Mais especificamente, tentou-se entender a capacidade e utilidade da ferramenta desenvolvida para checar QCs, sugerir axiomas através do Questionador e de suas operações de rastreamento. Além disso, com as respostas obtidas nos estudos de caso, hipóteses podem ser levantadas para investigações futuras, assim direcionando a comunidade para novas pesquisas, algo especialmente relevante para áreas novas como a rastreabilidade em engenharia de ontologias.

Para os estudos de caso, a questão inicialmente levantada foi dividida em quatro questões:

- Como o Verificador facilita o desenvolvimento, aumenta a produtividade de engenheiros e aumenta a qualidade de ontologias produzidas?
- Como o Questionador facilita o desenvolvimento, aumenta a produtividade de engenheiros e aumenta a qualidade de ontologias produzidas?
- Como o Rastreador facilita o desenvolvimento, aumenta a produtividade de engenheiros e aumenta a qualidade de ontologias produzidas?
- A ferramenta é útil e fácil de usar?

Desse modo, os resultados colhidos podem ser mais específicos e mais relevantes para investigação da capacidade do sistema em melhorar o desenvolvimento de ontologias.

O primeiro estudo de caso foi realizado por 7 pesquisadores com familiaridade em ontologias, sendo um doutor, quatro doutorandos e dois mestrandos na área. Eles efetuaram tarefas ligadas a cada um dos componentes (Verificador, Questionador e Rastreador) utilizando uma ontologia com informações sobre profissões criada especialmente para o estudo de caso. A lista de tarefas desse estudo de caso pode ser vista no Apêndice A.5.1.

No segundo estudo de caso, 3 alunos da disciplina Introdução à Informática em Saúde da UFPE, dois deles alunos dos cursos de computação e um do curso de medicina, efetuaram as tarefas listadas no Apêndice A.5.2 utilizando um módulo da SNOMED CT. Todo o conhecimento sobre ontologias desses alunos foi obtido na disciplina citada.

Após executar todas as tarefas, uma entrevista não estruturada foi realizada individualmente com cada participante. Apesar de não estruturada, um conjunto inicial de perguntas foi realizado com cada participante e, quando necessário, foram feitas mais perguntas para esclarecer alguma resposta ou para aprofundar algum assunto abordado. A Tabela 6.9 traz as perguntas iniciais realizadas com cada participante.

Tabela 6.9: Perguntas iniciais da entrevista não estruturada

#	Pergunta
1	Você acha que a checagem de Questões de Competência usando linguagem natural da ferramenta poderia facilitar o desenvolvimento de ontologias? Por quê?
2	Você acha que a checagem de Questões de Competência usando linguagem natural da ferramenta poderia aumentar a produtividade de um engenheiro de ontologias? Por quê?
3	Você acha que a checagem de Questões de Competência usando linguagem natural da ferramenta poderia aumentar a qualidade da ontologia produzida? Por quê?
4	Quais os pontos positivos e negativos da checagem de Questões Competência da ferramenta?
5	Você acha que a ferramenta fazer perguntas para ajudar um engenheiro de ontologias a adicionar axiomas poderia facilitar o desenvolvimento de ontologias? Por quê?
6	Você acha que a ferramenta fazer perguntas para ajudar um engenheiro de ontologias a adicionar axiomas poderia aumentar a produtividade do engenheiro? Por quê?
7	Você acha que a ferramenta fazer perguntas para ajudar um engenheiro de ontologias a adicionar axiomas poderia aumentar a qualidade da ontologia produzida? Por quê?
8	Quais os pontos positivos e negativos das perguntas realizadas pela ferramenta para adicionar axiomas?
9	Você acha que as operações de rastreamento da ferramenta poderiam facilitar o desenvolvimento de ontologias? Por quê?
10	Você acha que as operações de rastreamento da ferramenta poderiam aumentar a produtividade de um engenheiro de ontologias? Por quê?
11	Você acha que as operações de rastreamento da ferramenta poderiam aumentar a qualidade da ontologia produzida? Por quê?
12	Quais os pontos positivos e negativos das operações de rastreamento da ferramenta?
13	Você achou a ferramenta fácil de usar? Por quê?
14	Você achou a ferramenta útil? Por quê?

Além da entrevista, também foram coletados logs do uso da ferramenta, que registram

cada interação feita pelos usuários. Portanto, as informações passadas na entrevista, se necessário, podem ser validadas através dos logs. Ter mais de uma fonte de informação em um estudo de caso traz maior confiabilidade para as conclusões constatadas.

6.4.5.1 Resultados e Discussão

As respostas obtidas nas entrevistas e os logs das interações dos usuários foram usados para inferir conclusões a respeito das quatro questões enumeradas nos estudos de caso.

Sobre o Verificador todos os 10 participantes destacaram que o uso de linguagem natural para checar questões de competência torna o uso da ferramenta fácil, inclusive, fazendo com que a checagem seja útil para pessoas sem muito conhecimento a respeito de ontologias, lógica de descrição e OWL. Um participante mencionou que alguém leigo em relação a ontologias pode ser especialista em um domínio que está sendo modelado, assim, incluir este tipo de pessoa no processo de desenvolvimento de ontologias pode agregar um grande valor ao artefato produzido.

A checagem também ataca diretamente tarefas de verificação e validação de ontologias, procedimentos essenciais na engenharia de ontologias. As questões de competência podem ser usadas para diminuir a quantidade de erros e encontrar inconsistências. Três participantes apontaram essas características durante as entrevistas.

É importante destacar que a checagem de questões de competência foi frequentemente abordada como uma das melhores funcionalidades da ferramenta. Apenas quatro participantes entrevistados mencionaram algum ponto negativo, porém todos eles reconheceram que a checagem de questões usando linguagem natural pode influenciar diretamente no desenvolvimento de uma ontologia, tanto melhorando sua qualidade, quanto a produtividade do engenheiro.

Como pontos negativos, um participante comentou que apesar de ser fácil interagir usando linguagem natural, criar perguntas para testar pontos específicos de uma ontologia pode não ser simples em alguns casos. Outro ponto relevante apontado por um participante foi que as perguntas não levam em consideração o engajamento ontológico da questão, permitindo que o usuário faça perguntas que não façam sentido, mesmo sendo logicamente corretas.

Cinco participantes dos estudos de caso afirmaram explicitamente que a ferramenta é útil para leigos, entretanto, um deles argumentou que ela talvez possa ser preterida por engenheiros mais experientes, que já estão acostumados com outras ferramentas mais conhecidas, por exemplo, o Protégé. Esse comentário reforça a discussão a respeito da integração das funcionalidades da ferramenta com outros sistemas mais conhecidos. Como o método proposto nessa tese está alinhado a metodologias iterativas e que usam questões de competência para especificar requisitos, a integração das funcionalidades parece ser uma excelente opção para atender tanto usuários leigos quanto experientes.

Por fim, um participante fez uma observação sobre a suposição de mundo aberto e a suposição de mundo fechado. As ontologias trabalhadas usam a suposição de mundo aberto, que determina que mesmo que um axioma não esteja especificado na ontologia, ele ainda pode ser verdadeiro. Se ele não está na ontologia, ele apenas não é conhecido ou não foi explicitado.

Assim, quando uma QC é realizada com resposta esperada “true” e a ontologia não tem o conhecimento para respondê-la como verdadeira, o sistema não deveria mostrar que a resposta da QC é falsa, mas apenas que a ontologia não tem o conhecimento necessário. É uma diferença sutil, porém importante.

Sobre o Questionador seis participantes dos estudos de caso destacaram a facilidade de inserir axiomas respondendo perguntas como um ponto positivo da ferramenta. A sintaxe da OWL, a notação da DL ou até mesmo as operações de outras ferramentas para adicionar axiomas manualmente podem ser complicadas, principalmente para usuários leigos, assim tornar esse processo mais simples pode ter um impacto direto na produtividade das pessoas envolvidas no desenvolvimento de uma ontologia.

As funcionalidades providas pelo Questionador também foram elogiadas por um participante por se antecipar ao engenheiro sugerindo axiomas para agilizar o processo e tornar o desenvolvimento mais fácil. Um dos pontos fortes dessa abordagem apontado por outro participante é que o componente procura as lacunas no que foi especificado na ontologia e sugere para os engenheiros como preenchê-las. O componente também tem o potencial de melhorar o processo de desenvolvimento ajudando a resolver impasses com suas sugestões através das perguntas criadas automaticamente.

O último ponto positivo comentado nas respostas da entrevista sobre o Questionador é que as perguntas geradas são feitas dentro de um contexto, ou seja, elas são feitas de acordo com a QC perguntada inicialmente. Isso foi destacado como uma boa forma de interagir com a ferramenta por um participante, o que está alinhado com a ideia de tornar a interação próxima a um diálogo.

O componente Questionador também recebeu críticas durante as entrevistas. Cinco participantes mencionaram que as perguntas geradas podem ser muito abstratas e em certos casos elas podem não fazer sentido. Isto acontece principalmente quando perguntas muito longas são geradas, pois elas representam axiomas grandes. Outro motivo é que o componente não considera o engajamento ontológico, podendo assim existir perguntas desconectadas com a realidade, apesar de logicamente corretas. Outro ponto negativo reportado foi que as QSs precisam ser analisadas antes que o usuário permita o sistema adicionar aquele conhecimento, portanto, não se deve aceitar sempre o que o sistema sugere.

Sobre o Rastreador os 10 participantes dos estudos de caso destacaram que o componente traz informações importantes relacionadas a como a ontologia foi construída. Através das operações de rastreamento é possível entender de onde o sistema obteve informações para responder as perguntas e quais axiomas correspondem a cada pergunta. Adicionalmente, é possível analisar a abrangência de uma QC, verificando quais conceitos e axiomas estão relacionados a ela.

O rastreamento deixa claro para os usuários o que está acontecendo durante o desenvolvimento e como as decisões foram tomadas pelo sistema. Seis participantes comentaram que é útil para um engenheiro entender o motivo de um axioma ser adicionado e do resultado das checagens das QCs. Também foi apontada por dois participantes a capacidade do rastreamento

em ensinar aos usuários, pois, mostrando como as decisões foram tomadas pelo sistema e quais axiomas foram adicionados para satisfazer as QCs, um engenheiro pode aprender como tomar decisões futuras sobre a modelagem de ontologias.

Por fim, também foi discutido por um participante que o Rastreador serve de memória auxiliar que armazena tudo que foi feito usando a ferramenta, sendo útil para que os engenheiros recuperem informações importantes e analisem o processo de desenvolvimento.

Como pontos negativos, três participantes mencionaram que não é clara a influência do Rastreador na melhoria da produtividade ou no aumento da facilidade de desenvolvimento de uma ontologia. Além disso, um participante mencionou que a quantidade de informações gravadas e apresentadas para o usuário da ferramenta pode exigir um esforço grande para ser interpretada, assim prejudicando mais que ajudando. Dessa forma, foi sugerido que a interface do rastreamento poderia ser melhorada através de filtros e outras funcionalidades que facilitem a navegação pelos rastros gravados.

Sobre a utilidade e facilidade de uso da ferramenta, os 10 participantes destacaram que a ferramenta é fácil de usar e que sua interface simples é um dos principais motivos para isso. Outro ponto importante para facilidade de uso é a utilização da linguagem natural, pois evitar a sintaxe da DL e da OWL ajuda a tornar a interação com a ferramenta simples e ainda auxilia usuários leigos. A gravação e recuperação de informações pelo Rastreador e a geração de perguntas pelo Questionador tiveram sua utilidade reconhecida, apesar de ressalvas. Entretanto, o destaque da ferramenta, de acordo com as entrevistas, é o componente de checagem. Todos os participantes perceberam o potencial do componente e incluíram ele nos principais elogios.

Como ponto negativo, um comentário foi recorrente durante as entrevistas. A qualidade das perguntas geradas pelo Questionador foi apontada como um problema em alguns casos. Elas podem ser complicadas de analisar ou podem não fazer sentido, diminuindo a utilidade das perguntas sugeridas, pois um engenheiro passaria por dificuldades para avaliar se a pergunta deve ser respondida positivamente ou não. Dois participantes também criticaram a exibição das informações pelo Rastreador, que pode ser difícil de ser navegada, diminuindo tanto a facilidade de uso, quanto a utilidade desse componente.

Para finalizar a discussão, as Tabela 6.10 e 6.11 trazem um resumo das respostas dos participantes sobre os componentes da ferramenta. Na Tabela 6.10, para cada componente foi verificado o número de participantes que responderam sim, não ou talvez para a sua influência em cada uma das características analisadas nas entrevistas (facilidade no desenvolvimento, produtividade do engenheiro e qualidade da ontologia). Sim significa que o participante se mostrou certo da influência do componente na característica analisada; não significa que ele acha que o componente não influencia na característica; talvez foi atribuído para os casos incertos, quando o participante fica em dúvida se o componente influência ou não em uma característica.

A Tabela 6.11 mostra os principais pontos positivos e negativos abordados pelos participantes dos estudos de caso. Cada linha da tabela apresenta um dos componentes analisados e as colunas trazem os pontos positivos e negativos respectivamente.

Tabela 6.10: Resumo da opinião dos entrevistados sobre a capacidade de cada componente em facilitar o desenvolvimento, aumentar a produtividade de um engenheiro e aumentar a qualidade de ontologias produzidas

	Verificador			Questionador			Rastreador		
	Sim	Não	Talvez	Sim	Não	Talvez	Sim	Não	Talvez
Facilidade no desenvolvimento	9	0	1	10	0	0	6	2	2
Produtividade do engenheiro	9	0	1	10	0	0	6	2	2
Qualidade da ontologia	10	0	0	8	0	2	9	1	0

Tabela 6.11: Resumo dos pontos positivos e negativos sobre os componentes da ferramenta

	Pontos positivos	Pontos negativos
Verificador	Linguagem natural facilita o uso; Útil para leigos; Ajuda a diminuir erros e inconsistências;	Não possui engajamento ontológico;
Questionador	Simplifica a adição de axiomas usando perguntas e respostas; Sugere axiomas; Procura lacunas na ontologia; Perguntas realizadas dentro de um contexto.	Perguntas geradas podem não fazer sentido; É necessário analisar as perguntas sugeridas.
Rastreador	Traz informações importantes sobre o desenvolvimento; Deixa claro as razões de decisões tomadas pela ferramenta; Serve de memória do projeto.	Não é clara a influência na produtividade; Em grande quantidade, as informações são difíceis de analisar.

6.5 Conclusão

Este capítulo apresentou experimentos realizados para avaliar o método de expansão de ontologias e a ferramenta desenvolvida. Mais especificamente foram executados testes para avaliar os componentes Verificador e Questionador, e, por fim, foram realizados dois estudos de caso para avaliar toda a ferramenta, abrangendo todos os componentes, incluindo o Rastreador.

Através da checagem provida pelo componente Verificador foi mostrado que é possível checar os axiomas presentes em uma ontologia em DL $\mathcal{EL}$ e checar perguntas que precisam de inferência para serem respondidas. Todos os casos testados foram checados com sucesso pelo componente.

De três testes realizados, o Questionador conseguiu, respectivamente, gerar perguntas que ao serem respondidas adicionaram conhecimento à ontologia para satisfazer uma QC, em 60,32%, 66,67% e 80,30% dos casos. Além disso, foi analisado se os axiomas adicionados pelo Questionador eram parecidos com axiomas codificados por engenheiros. O componente conseguiu uma taxa de precisão maior que a de abrangência nos testes realizados, o que evidencia que ele consegue sugerir axiomas que possuem relação com os axiomas dos engenheiros, mas, frequentemente, ele sugere axiomas incompletos quando comparados com os axiomas dos engenheiros.

Nos estudos de caso, foram realizados testes da ferramenta com 10 participantes, em seguida, foram efetuadas entrevistas para colher a opinião deles sobre a facilidade de desenvolvimento, a produtividade e qualidade dos artefatos produzidos usando a ferramenta. A checagem de QCs foi o ponto de maior destaque, tendo o potencial de facilitar o desenvolvimento e de melhorar a produtividade e a qualidade dos artefatos. Adicionalmente, foi destacado que a checagem usando linguagem natural facilita o uso da ferramenta por pessoas leigas em ontologias. A adição de axiomas usando o Questionador também foi elogiada e seu potencial reconhecido, entretanto foram feitas ressalvas quanto a qualidade das perguntas geradas. Por fim, o Rastreador foi avaliado como importante para entender como a ferramenta está ajudando a construir a ontologia, assim como servir de memória do projeto para análises durante o desenvolvimento, por outro lado, alguns entrevistados apontaram que o rastreamento pode não ter relação com a facilidade de desenvolvimento e a produtividade.

O próximo capítulo traz as conclusões gerais do trabalho apresentado, mostrando as contribuições realizadas e discussões sobre as limitações do que foi proposto e desenvolvido para guiar futuras pesquisas relacionadas ao tema da tese.

7

Conclusões

Esta tese de doutorado teve como objetivo criar um método, com uma ferramenta de suporte, de expansão de ontologias focado em questões de competência que possua um sistema de rastreabilidade automática de requisitos. Dessa forma, avançando e melhorando áreas relacionadas à engenharia de ontologias com potencial de aumentar a qualidade dos artefatos produzidos, aumentar a produtividade dos engenheiros e facilitar o processo de desenvolvimento.

Através dos experimentos apresentados no Capítulo 6 foi analisado se o sistema e o método proposto conseguiram atingir os objetivos almejados, além de ter sido verificado através de dois estudos de caso se a implementação do sistema tem a capacidade de melhorar o processo de desenvolvimento de ontologias.

Os experimentos foram realizados usando a ontologia SNOMED CT. Seus resultados mostraram que o sistema e o método são capazes de checar QCs que representam axiomas em uma ontologia, inclusive QCs que representam axiomas inferidos. Além disso, o sistema consegue sugerir axiomas que ao serem adicionados a uma ontologia fazem uma QC não satisfeita ser satisfeita. Para isso, o sistema transforma o axioma sugerido em uma representação em linguagem natural no formato de uma pergunta que é apresentada para o usuário responder. O sistema também consegue escrever axiomas em ontologias e prover uma série de operações para um engenheiro visualizar os rastros do desenvolvimento usando o sistema. Todas essas funcionalidades são suportadas para ontologias em DL $\mathcal{EL}$.

O sistema mostra o potencial de novas abordagens para engenharia de ontologias. A checagem de questões de competência usando linguagem natural automatiza o processo de checagem dos requisitos, evitando que engenheiros precisem checá-los manualmente, e abstrai a notação complexa da DL ou da OWL. Adicionalmente, a sugestão de axiomas através de perguntas, tem como objetivo guiar o engenheiro a adicionar axiomas para preencher lacunas no conhecimento representado pela ontologia. Por fim, o sistema traz uma implementação de rastreabilidade automática de requisitos, que mostra as possibilidades de gravação de rastros sem intervenção direta do engenheiro, de como os rastros podem ser navegados, e introduz de forma explícita essas atividades à engenharia de ontologias.

Também foram executados dois estudos de caso que são apresentados no Capítulo 6. O

primeiro estudo de caso foi realizado com 7 participantes pesquisadores na área de ontologias. O segundo estudo de caso foi realizado com 3 participantes que possuíam conhecimento básico na área de ontologias, mas também possuíam conhecimento na área biológica e em computação. Através deles foi concluído que a checagem de QCs é o ponto mais forte da ferramenta, podendo facilitar o desenvolvimento de ontologias e incluir pessoas leigas no processo de desenvolvimento. A geração de perguntas também se mostrou útil, porém a qualidade das perguntas geradas poderia melhorar. O rastreamento de requisitos ajuda a deixar claro as decisões tomadas pela ferramenta e serve como memória do projeto, trazendo a garantia de que as questões de competência estão sendo satisfeitas. O rastreamento mostra ainda as dependências de questões em relação a axiomas, o que é muito importante no caso de se alterar a ontologia, já que evidencia que certos axiomas não podem ser retirados, sob pena de a ontologia não preencher os requisitos que lhe foram imputados.

7.1 Contribuições Realizadas

Através da pesquisa apresentada nessa tese, tem-se que as principais contribuições realizadas foram:

A criação de um método de expansão de ontologias com foco em questões de competência. Neste trabalho foi definido como uma ontologia pode ser expandida através de uma abordagem que utiliza perguntas (questões de competência) e respostas. Tanto o usuário pode fornecer perguntas para um sistema implementando o método, quanto o sistema pode perguntar e esperar respostas do usuário para adicionar novos conhecimentos a uma ontologia. O método tem uma natureza iterativa, na qual o engenheiro de ontologias vai expandindo a ontologia passo a passo, perguntando e respondendo quando necessário. Também foi definida uma arquitetura para um sistema que implemente o método, quais são os seus componentes principais, quais suas responsabilidades e como eles interagem uns com os outros. Este método facilita a interação dos engenheiros para checar QCs e adicionar axiomas. Para isso usa-se linguagem natural, que diminui as barreiras para o desenvolvimento, pois não é necessário se preocupar com notações em DL ou OWL. Isto pode atrair pessoas leigas em ontologias, mas que sejam especialistas no domínio representado pela ontologia.

A construção de uma ferramenta que implementa o método definido. Além de propor o método e a arquitetura para implementação dele, também foi criada uma ferramenta seguindo a especificação do método. A ferramenta possui todas as funcionalidades do método e foi disponibilizada no endereço: https://yurimalheiros.github.io/ionscq/. Com isso, engenheiros de ontologias de todo o mundo podem utilizar as inovações criadas por esse trabalho, como a checagem automática, a sugestão de axiomas e a rastreabilidade de requisitos.

A implementação de um sistema de checagem automática de QCs usando linguagem natural controlada. A ferramenta possui um sistema de checagem automática de QCs que usa linguagem natural controlada. Assim, os engenheiros podem checar QCs na ontologia

sem precisar usar notações de lógica de descrição. É importante ressaltar que além de usar os axiomas codificados explicitamente na ontologia, o componente utiliza um raciocinador para inferir novas relações que podem ser usadas para responder as QCs. Esta foi a funcionalidade de maior destaque apontada pelos participantes dos estudos de caso, pois checar requisitos usando perguntas é uma forma fácil e natural para se interagir com as ontologias.

A implementação de um sistema de rastreabilidade automática de requisitos para ontologias. A ferramenta também fornece um sistema de rastreabilidade automática de requisitos, que vai gravando todo o rastro deixado pelos engenheiros ao usar o sistema. Com ele é possível verificar todas as perguntas e respostas feitas pelo engenheiro e também as criadas pelo sistema. Além disso, o sistema de rastreabilidade traz a funcionalidade para verificar o que foi alterado quando o usuário responde uma questão feita pelo sistema, quais conceitos são impactados em cada pergunta e resposta e quais perguntas estão relacionadas com uma determinada classe. Tem-se assim rastros nas duas vias, gravados automaticamente, das QCs para os axiomas e das classes para as QCs. Tal abordagem, baseada em lógica e raciocínio automático, além de rara na engenharia de ontologias, é difícil de replicar completamente na engenharia de software, devido a natureza dos seus artefatos e da representação dos requisitos.

O uso de unificação para sugerir axiomas auxiliando o desenvolvimento de ontologias. Para sugerir axiomas que podem fazer um QC não satisfeita ser satisfeita, foi utilizada uma técnica que usa unificação em lógica de descrição (Baader & Morawska, 2009). Com o resultado da unificação o sistema transforma os axiomas sugeridos em perguntas em linguagem natural, para conseguir auxiliar engenheiros a expandirem ontologias usando o método de perguntas e respostas. Assim, o engenheiro é guiado para que ele consiga preencher lacunas no conhecimento representado pela ontologia, sem precisar codificar os axiomas usando DL ou OWL.

7.2 Trabalhos Futuros e Limitações

Com o término do trabalho apresentado na tese foram levantados pontos que podem ser melhorados para aperfeiçoar os resultados obtidos e para que a pesquisa aqui iniciada possa ser continuada. A seguir são mostradas limitações e ideias para trabalhos futuros.

Suportar mais perguntas em linguagem natural no Verificador. A interação com sistemas através de linguagem natural, especialmente através de perguntas e respostas, é um problema pesquisado pela comunidade de Inteligência Artificial por anos. O componente Verificador apresentado suporta um número limitado de perguntas e respostas, que, apesar de cobrir todos os operadores da DL $\mathcal{EL}$ e conseguir checar QCs em uma grande ontologia como a SNOMED, ainda poderia evoluir. Com os avanços na área de processamento de linguagem natural, o componente de checagem de questões de competência pode ganhar muito em facilidade de uso, pois ele entenderia mais tipos de perguntas. Assim, cada vez mais a necessidade do entendimento da lógica de descrição poderia ser deixado de lado para checar QCs em uma ontologia, fazendo com que a consulta de QCs em uma ontologia seja acessível para os usuários

mais leigos.

Melhorar o desempenho da ferramenta. Durante o desenvolvimento da ferramenta, a velocidade de processamento para entender perguntas, fornecer respostas e etc., não foi foco de grandes otimizações. Com isso, algumas operações no sistema podem ser melhoradas para responder mais rapidamente aos comandos do usuário. Nesse quesito, destaca-se as operações de checagem e, principalmente, as operações que geram perguntas usando unificação. A melhoria da velocidade dessas operações traria uma experiência melhor no uso da ferramenta e consequentemente aumentaria a utilidade do sistema para os usuários.

Investigar outras técnicas além da unificação. Em alguns casos mostrados nos experimentos, a unificação não consegue sugerir axiomas para o usuário, além disso, nos estudos de caso, os participantes apontaram que a qualidade das perguntas geradas podem ser um problema para os usuários. Com isso, outras técnicas para sugerir axiomas, que tornarão um axioma antes não deduzido em dedutível, merecem mais investigação, pois elas podem trazer resultados melhores para o sistema. A abdução em DL (Halland & Britz, 2012), por exemplo, pode ser usada para o mesmo propósito que a unificação foi utilizada nesse trabalho, fazendo com que ela seja uma técnica a ser investigada no futuro.

Suportar refatoração. Após adicionar axiomas utilizando o sistema pode ser necessário realizar refatoração para melhorar e organizar a estrutura da ontologia. Suportar esse tipo de atividade traz mais opções para um engenheiro trabalhar com as representações codificadas, além de permitir mudanças em definições já existentes.

Explicações de axiomas. Explicações de axiomas (*axiom pinpointing*) (Baader & Peñaloza, 2007) é um serviço de inferência em DL que provê uma justificativa para uma implicação lógica. Assim, usar explicações ligadas às QCs traz informações adicionais para os engenheiros entenderem as razões de uma QC está sendo satisfeita. Tal operação poderia ser vinculada ao componente Rastreador responsável pela rastreabilidade automática.

Aprofundar os estudos sobre a rastreabilidade em engenharia de ontologias. Este trabalho apresentou um sistema de rastreabilidade automática de requisitos para ontologias, área muitas vezes esquecida nos trabalhos de engenharia de ontologias. Como foi visto que é possível gravar os rastros e prover ferramentas para os engenheiros de ontologias navegarem através dessa informação, um próximo passo, que não foi realizado nesse trabalho, é aprofundar os estudos quantitativos e qualitativos da rastreabilidade. Assim, espera-se ganhar mais conhecimento sobre que operações são essenciais para a rastreabilidade em ontologias, o que pode ser melhorado no que já foi desenvolvido, se existem novas operações que engenheiros necessitam, e, por fim, comparar mais a área de rastreabilidade em engenharia de ontologias com a de rastreabilidade em engenharia de software.

Incorporar funcionalidades em outras ferramentas. O método de expansão focado em questões de competência está alinhado a outras metodologias propostas para o desenvolvimento de ontologias. Assim, as ideias do método proposto, podem ser incorporadas a processos já existentes. Para isso, um trabalho futuro que pode ser realizado é a incorporação dos com-

ponentes do sistema desenvolvido em ferramentas já existentes, por exemplo, o Protégé, que é amplamente usando pela comunidade e possui um sistema de plugins que permite a adição de novos recursos por terceiros. Com isso, o método proposto estaria disponível para mais engenheiros de ontologias e, não seria necessário, usar uma ferramenta diferente para aproveitar os recursos de checagem usando linguagem natural, geração de perguntas e rastreabilidade automática de requisitos.

Referências

Antoniou, Grigoris, & Van Harmelen, Frank. 2004. *A Semantic Web Primer*. MIT press.

Arpírez, Julio C., Corcho, Oscar, Fernández-López, Mariano, & Gómez-Pérez, Asunción. 2001. WebODE: a scalable workbench for ontological engineering. *Pages 6–13 of: Proceedings of the 1st International Conference on Knowledge Capture*. New York, NY, USA: ACM.

Asuncion, Hazeline U, François, Frédéric, & Taylor, Richard N. 2007. An end-to-end industrial software traceability tool. *Pages 115–124 of: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM.

Auer, Sören, & Herre, Heinrich. 2006. RapidOWL—An agile knowledge engineering methodology. *Pages 424–430 of: International Andrei Ershov Memorial Conference on Perspectives of System Informatics*. Springer.

Aurum, Aybüke, & Wohlin, Claes. 2005. *Engineering and Managing Software Requirements*. Vol. 1. Springer.

Baader, Franz. 2003. *The description logic handbook: theory, implementation, and applications*. Cambridge university press.

Baader, Franz, & Morawska, Barbara. 2009. Unification in the Description Logic $\mathcal{EL}$. *Pages 350–364 of: Rewriting techniques and applications*. Springer.

Baader, Franz, & Morawska, Barbara. 2010. SAT Encoding of Unification in $\mathcal{EL}$. *Pages 97–111 of: Logic for Programming, Artificial Intelligence, and Reasoning*. Springer.

Baader, Franz, & Peñaloza, Rafael. 2007. Axiom pinpointing in general tableaux. *Pages 11–27 of: International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*. Springer.

Baader, Franz, Borgwardt, Stefan, Mendez, Julian Alfredo, & Morawska, Barbara. 2012. UEL: Unification solver for $\mathcal{EL}$. *Pages 26–36 of: Proc. of the 25th Int. Workshop on Description Logics (DL'12)*, vol. 846. Citeseer.

Beck, Kent. 2000. *Extreme programming explained: embrace change*. addison-wesley professional.

Beck, Kent. 2003. *Test-driven development: by example*. Addison-Wesley Professional.

Berners-Lee, Tim, Hendler, James, Lassila, Ora, *et al.* . 2001. The semantic web. *Scientific american*, **284**(5), 28–37.

Bezerra, Camila, Freitas, Fred, & Santana, Filipe. 2013. Evaluating ontologies with competency questions. *Pages 284–285 of: Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2013 IEEE/WIC/ACM International Joint Conferences on*, vol. 3. IEEE.

Buchheit, Martin, Donini, Francesco M, & Schaerf, Andrea. 1993. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*.

Cleland-Huang, Jane, Settimi, Raffaella, BenKhadra, Oussama, Berezhanskaya, Eugenia, & Christina, Selvia. 2005. Goal-centric traceability for managing non-functional requirements. *Pages 362–371 of: Proceedings of the 27th international conference on Software engineering*. ACM.

Cook, Stephen A. 1971. The complexity of theorem-proving procedures. *Pages 151–158 of: Proceedings of the third annual ACM symposium on Theory of computing*. ACM.

Copeland, Maria, Brown, Andy, Parkinson, Helen E, Stevens, Robert, & Malone, James. 2012. The SWO Project: A Case Study for Applying Agile Ontology Engineering Methods for Community Driven Ontologies. *ICBO*, **7**, 2012.

Cuenca Grau, Bernardo, Horrocks, Ian, Kazakov, Yevgeny, & Sattler, Ulrike. 2008. Modular reuse of ontologies: Theory and practice. *Journal of Artificial Intelligence Research*, **31**, 273–318.

De Lucia, Andrea, Oliveto, Rocco, & Tortora, Genoveffa. 2008. Adams re-trace: traceability link recovery via latent semantic indexing. *Pages 839–842 of: Proceedings of the 30th international conference on Software engineering*. ACM.

Dean, Mike, Schreiber, Guus, Bechhofer, Sean, van Harmelen, Frank, Hendler, Jim, Horrocks, Ian, McGuinness, Deborah L, Patel-Schneider, Peter F, & Stein, L Andrea. 2004. OWL web ontology language reference. *W3C Recommendation February*, **10**.

del Carmen Suárez-Figueroa, María, de Cea, Guadalupe Aguado, Buil, Carlos, Dellschaft, Klaas, Fernández-López, Mariano, García, Andrés, Gómez-Pérez, Asunción, Herrero, German, Montiel-Ponsoda, Elena, Sabou, Marta, Villazon-Terrazas, Boris, & Yufei, Zheng. 2008 (Feb.). *NeOn Methodology for Building Contextualized Ontology Networks*.

Del Vescovo, Chiara, Parsia, Bijan, Sattler, Ulrike, & Schneider, Thomas. 2010. The modular structure of an ontology: an empirical study. *Description Logics*, **573**.

Devedzić, Vladan. 2002. Understanding ontological engineering. *Communications of the ACM*, **45**(4), 136–144.

Egyed, Alexander, & Grünbacher, Paul. 2005. Supporting software understanding with automated requirements traceability. *International Journal of Software Engineering and Knowledge Engineering*, **15**(05), 783–810.

Elsenbroich, Corinna, Kutz, Oliver, & Sattler, Ulrike. 2006. A Case for Abductive Reasoning over Ontologies. *In: OWLED*, vol. 216.

Farquhar, Adam, Fikes, Richard, Pratt, Wanda, & Rice, James. 1995. *Collaborative ontology construction for information integration*. Tech. rept. Technical Report KSL-95-63, Stanford University Knowledge Systems Laboratory.

Fernandez-Lopez, Mariano, Gomez-Perez, Asuncion, & Juristo, Natalia. 1997 (March). METHONTOLOGY: from Ontological Art towards Ontological Engineering. *Pages 33–40 of: Proceedings of the AAAI97 Spring Symposium*.

Gangemi, Aldo, & Presutti, Valentina. 2009. Ontology design patterns. *Pages 221–243 of: Handbook on ontologies*. Springer.

Gennari, John H, Musen, Mark A, Fergerson, Ray W, Grosso, William E, Crubézy, Monica, Eriksson, Henrik, Noy, Natalya F, & Tu, Samson W. 2003. The evolution of Protégé: an environment for knowledge-based systems development. *International Journal of Human-computer studies*, **58**(1), 89–123.

Gervasi, Vincenzo, & Zowghi, Didar. 2005. Reasoning about inconsistencies in natural language requirements. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, **14**(3), 277–330.

Gómez-Pérez, A., Fernández-López, M., & Corcho, O. 2004. *Ontological Engineering: With Examples from the Areas of Knowledge Management, E-Commerce and the Semantic Web*. Advanced Information and Knowledge Processing. Springer.

Gotel, Orlena CZ, & Finkelstein, Anthony CW. 1994. An analysis of the requirements traceability problem. *Pages 94–101 of: Requirements Engineering, 1994., Proceedings of the First International Conference on*. IEEE.

Grenning, James. 2012. Planning poker or how to avoid analysis paralysis while release planning.–2002. *Online: http://renaissancesoftware. net/files/articles/PlanningPoker-v1*, **1**.

Gruber, Thomas R. 1992. *Ontolingua: A mechanism to support portable ontologies*. Stanford University, Knowledge Systems Laboratory.

Gruber, Thomas R. 1995. Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies*, **43**(5), 907–928.

Gruninger, Michael, & Fox, Mark S. 1995. Methodology for the Design and Evaluation of Ontologies.

Guarino, Nicola, & Welty, Christopher. 2002. Evaluating ontological decisions with OntoClean. *Communications of the ACM*, **45**(2), 61–65.

Halland, Ken, & Britz, Katarina. 2012. ABox abduction in ALC using a DL tableau. *Pages 51–58 of: Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference*. ACM.

Hammond, Peter, & Sergot, Marek. 1984. APES: augmented prolog for expert systems. *Logic Based Systems Ltd*, **40**.

Hayes, Jane Huffman, Dekhtyar, Alex, Sundaram, Senthil Karthikeyan, Holbrook, E Ashlee, Vadlamudi, Sravanthi, & April, Alain. 2007. REquirements TRacing On target (RETRO): improving software maintenance through traceability recovery. *Innovations in Systems and Software Engineering*, **3**(3), 193–202.

Horridge, Matthew, & Bechhofer, Sean. 2011. The OWL API: A Java API for OWL ontologies. *Semantic Web*, **2**(1), 11–21.

Kazakov, Yevgeny, Krötzsch, Markus, & Simančík, František. 2011. Concurrent Classification of $\mathcal{EL}$ Ontologies. *Pages 305–320 of: The Semantic Web–ISWC 2011*. Springer.

Kazakov, Yevgeny, Krötzsch, Markus, & Simancik, Frantisek. 2012. ELK Reasoner: Architecture and Evaluation. *In: ORE*.

Knublauch, Holger. 2002. *An agile development methodology for knowledge-based systems including a Java framework for knowledge modeling and appropriate tool support*. Ph.D. thesis, Universität Ulm.

Koivunen, Marja Riitta, & Miller, Eric. 2001. W3c semantic web activity. *Semantic Web Kick-Off in Finland*, 27–44.

Lee, Dennis, de Keizer, Nicolette, Lau, Francis, & Cornet, Ronald. 2014. Literature review of SNOMED CT use. *Journal of the American Medical Informatics Association*, **21**(e1), e11–e19.

Lemaignan, Severin, Siadat, Ali, Dantan, J-Y, & Semenenko, Anatoli. 2006. MASON: A proposal for an ontology of manufacturing domain. *Pages 195–200 of: Distributed Intelligent Systems: Collective Intelligence and Its Applications, 2006. DIS 2006. IEEE Workshop on*. IEEE.

Lenat, Douglas B. 1995. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, **38**(11), 33–38.

Leuf, Bo, & Cunningham, Ward. 2001. The Wiki way: collaboration and sharing on the Internet.

Lin, Jun, Lin, Chan Chou, Cleland-Huang, Jane, Settimi, Raffaella, Amaya, Joseph, Bedford, Grace, Berenbach, Brian, Khadra, Oussama Ben, Duan, Chuan, & Zou, Xuchang. 2006. Poirot: A distributed tool supporting enterprise-wide automated traceability. *Pages 363–364 of: Requirements Engineering, 14th IEEE International Conference*. IEEE.

Malheiros, Yuri, & Freitas, Fred. 2014. A Method to Develop Description Logic Ontologies Iteratively with Automatic Requirement Traceability. *Pages 646–658 of: Informal Proceedings of the 27th International Workshop on Description Logics, Vienna, Austria, July 17-20, 2014.*

Manning, Christopher D., Surdeanu, Mihai, Bauer, John, Finkel, Jenny, Bethard, Steven J., & McClosky, David. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. *Pages 55–60 of: Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

McGuinness, Deborah L, & Van Harmelen, Frank. 2004. OWL web ontology language overview. *W3C recommendation*, **10**(10), 2004.

Mizoguchi, Riichiro. 1998. A step towards ontological engineering. *Pages 24–31 of: 12th National Conference on AI of JSAI*.

Motik, Boris, Grau, Bernardo Cuenca, Horrocks, Ian, Wu, Zhe, Fokoue, Achille, & Lutz, Carsten. 2009. OWL 2 Web Ontology Language: Profiles. *W3C recommendation*, **27**, 61.

Myers, Eugene W. 1986. An O(ND) Difference Algorithm and Its Variations. *Algorithmica*, **1**(1-4), 251–266.

Noy, Natalya F., & McGuinness, Deborah L. 2001 (March). *Ontology Development 101: A Guide to Creating Your First Ontology*. Tech. rept.

Pan, Jeff Z, Matentzoglu, Nico, Jay, Caroline, Vigo, Markel, & Zhao, Yuting. 2017. Understanding Author Intentions: Test Driven Knowledge Graph Construction. *Pages 1–26 of: Reasoning Web: Logical Foundation of Knowledge Graph Construction and Query Answering*. Springer.

Patel-Schneider, Peter F., Hayes, Patrick, & Horrocks, Ian. 2004 (February). *OWL Web Ontology Language Semantics and Abstract Syntax*. W3C Recommendation. W3C. Published online on February 10th, 2004 at http://www.w3.org/TR/2004/REC-owl-semantics-20040210/.

Pinheiro, Francisco AC. 2004. Requirements Traceability. *Pages 91–113 of: Perspectives on Software Requirements*. Springer.

Prud'Hommeaux, Eric, Seaborne, Andy, *et al.* . 2008. SPARQL query language for RDF. *W3C recommendation*, **15**.

Ramesh, Bala, Stubbs, Lt Curtis, Edwards, Michael, & Powers, Timothy. 1995. Lessons learned from implementing requirements traceability. *Crosstalk–Journal of Defense Software Engineering*, **8**(4), 11–15.

Rector, Alan L. 2003. Modularisation of domain ontologies implemented in description logics and related formalisms including OWL. *Pages 121–128 of: Proceedings of the 2nd international conference on Knowledge capture*. ACM.

Ren, Yuan, Parvizi, Artemis, Mellish, Chris, Pan, Jeff Z, Van Deemter, Kees, & Stevens, Robert. 2014. Towards competency question-driven ontology authoring. *Pages 752–767 of: European Semantic Web Conference*. Springer.

Rogers, J, & Rector, A. 1996. The GALEN ontology. *Medical Informatics Europe (MIE 96)*, 174–178.

Saiedian, Hossein, Kannenberg, Andrew, & Morozov, Serhiy. 2013. A streamlined, cost-effective database approach to manage requirements traceability. *Software Quality Journal*, **21**(1), 23–38.

Schmidt-Schauß, Manfred, & Smolka, Gert. 1991. Attributive concept descriptions with complements. *Artificial intelligence*, **48**(1), 1–26.

Sedano, FJ Farfán, Cuadrado, M Terrón, Rebolledo, EM García, Clemente, Yolanda Castellanos, Balazote, Pablo Serrano, & Delgado, Ángel Gómez. 2009. Implementation of SNOMED CT to the medicines database of a general hospital. *Studies in health technology and informatics*, **148**, 123–130.

Sergot, Marek. 1984. New Horizons in Educational Computing. New York, NY, USA: Halsted Press.

Skuce, Douglas. 1995. Conventions for reaching agreement on shared ontologies. *In: Proceedings of the 9th Knowledge Acquisition for Knowledge Based Systems Workshop*.

Spackman, Kent A, Dionne, Robert, Mays, Eric, & Weis, Jason. 2002. Role grouping as an extension to the description logic of Ontylog, motivated by concept modeling in SNOMED. *Page 712 of: Proceedings of the AMIA Symposium*. American Medical Informatics Association.

Staab, Steffen, Studer, Rudi, Schnurr, Hans-Peter, & Sure, York. 2001. Knowledge Processes and Ontologies. *IEEE Intelligent Systems*, **16**(1), 26–34.

Suárez-Figueroa, Mari Carmen. 2010. *NeOn Methodology for building ontology networks: specification, scheduling and reuse*. Ph.D. thesis, Informatica.

Suarez-Figueroa, Mari Carmen, Gomez-Perez, Asuncion, & Fernandez-Lopez, Mariano. 2012. The NeOn methodology for ontology engineering. *Pages 9–34 of: Ontology engineering in a networked world*. Springer.

Sure, York, Erdmann, Michael, Angele, Jürgen, Staab, Steffen, Studer, Rudi, & Wenke, Dirk. 2002. OntoEdit: Collaborative Ontology Development for the Semantic Web. *Pages 221–235 of: Proceedings of the First International Semantic Web Conference on The Semantic Web*. ISWC '02. London, UK, UK: Springer-Verlag.

Sure, York, Staab, Steffen, & Studer, Rudi. 2004. On-to-knowledge methodology (OTKM). *Pages 117–132 of: Handbook on ontologies*. Springer.

Uschold, Mike, & Gruninger, Michael. 1996. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, **11**, 93–136.

W3C. 2009. OWL 2 Web Ontology Language Document Overview.

Wiegers, K., & Beatty, J. 2013. *Software Requirements*. Developer Best Practices. Pearson Education.

Wieringa, Roel. 1995. *An Introduction to Requirements Traceability*.

Winkler, Stefan, & Pilgrim, Jens. 2010. A survey of traceability in requirements engineering and model-driven development. *Software and Systems Modeling (SoSyM)*, **9**(4), 529–565.

Winkler, William E. 1990. String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage. *Pages 354–359 of: Proceedings of the Section on Survey Research*.

Zemmouchi-Ghomari, Leila, & Ghomari, Abdessamed Réda. 2013. Translating Natural Language Competency Questions into SPARQLQueries: A Case Study. *Pages 81–86 of: WEB 2013, The First International Conference on Building and Exploring Web Based Environments*.

Apêndice

A

Dados dos Experimentos e Estudos de Caso

A.1 Axiomas Escolhidos para Testar o Verificador

$$AllergenClass \sqsubseteq Substance \tag{A.1}$$

$$DrugOrMedicament \sqsubseteq Substance \tag{A.2}$$

$$AminoAcidSupplement \sqsubseteq ProteinSupplementation \tag{A.3}$$

$$SynovialStructureOfLimb \sqsubseteq LimbStructure \sqcap RegionalSynovialStructure \tag{A.4}$$

$$\begin{aligned} CongenitalAnomalyOfBodyCavity \equiv\ & CongenitalAnomalyOfTrunk \sqcap Disease \\ & \sqcap ((\exists associatedMorphology.CongenitalAnomaly) \\ & \sqcap (\exists associatedMorphology.DevelopmentalAbnormality) \\ & \sqcap (\exists findingSite.BodyCavityStructure) \\ & \sqcap (\exists occurrence.Congenital)) \\ & \sqcap (\exists occurrence.Congenital) \end{aligned} \tag{A.5}$$

$$Dysplasia \sqsubseteq GrowthAlteration \sqcap Lesion \tag{A.6}$$

$$ProcedureOnSkeletalSystem \equiv ProcedureOnMusculoskeletalSystem \sqcap (\exists procedureSite.SkeletalSystemStructure) \quad \text{(A.7)}$$

$$AbdomenDestructiveProcedure \equiv DestructiveProcedure \sqcap ProcedureOnAbdomen \sqcap ((\exists method.DestructionAction) \sqcap (\exists procedureSiteDirect.AbdominalStructure)) \quad \text{(A.8)}$$

$$SerousSacStructure \sqsubseteq BodyOrganStructure \quad \text{(A.9)}$$

$$LegDestructiveProcedure \equiv DestructiveProcedure \sqcap ((\exists method.DestructionAction) \sqcap (\exists procedureSiteDirect.LowerLegStructure)) \quad \text{(A.10)}$$

$$FindingOfRegionOfThorax \equiv FindingOfTrunkStructure \sqcap (\exists findingSite.ThoracicStructure) \quad \text{(A.11)}$$

$$DrugAllergenOrPseudoallergen \sqsubseteq AllergenOrPseudoallergen \quad \text{(A.12)}$$

$$AbdominalCavityStructure \sqsubseteq AbdominalStructure \sqcap BodyCavityStructure \sqcap StructureOfBodyCavitySubdivision \sqcap StructureOfCompartmentOfAbdomen \quad \text{(A.13)}$$

$$DisorderOfBone \equiv DisorderOfSkeletalSystem \sqcap (\exists findingSite.BoneStructure) \quad \text{(A.14)}$$

$$LowerTrunkStructure \sqsubseteq LowerBodyPartStructure \sqcap StructureOfSubregionOfTrunk \quad \text{(A.15)}$$

$$RightLowerExtremityStructure \equiv LowerLimbStructure \sqcap StructureOfRightHalfOfBody \quad \text{(A.16)}$$

$$StructureOfViscus \sqsubseteq BodyOrganStructure \quad \text{(A.17)}$$

$$DestructionAction \sqsubseteq SurgicalAction \quad \text{(A.18)}$$

$$SurgicalProcedureOnThorax \equiv ProcedureOnThorax \sqcap SurgicalProcedure \sqcap ((\exists method.SurgicalAction) \sqcap (\exists procedureSite.ThoracicStructure)) \quad \text{(A.19)}$$

$$Fixation \equiv ProcedureByMethod \sqcap (\exists method.FixationAction) \quad \text{(A.20)}$$

$$MorphologicallyAlteredStructure \sqsubseteq BodyStructure \quad \text{(A.21)}$$

$$TraumaticInjuryOfExternalGenitalia \equiv GenitalInjury \sqcap ((\exists associatedMorphology.TraumaticAbnormality) \sqcap (\exists findingSite.ExternalGenitaliaStructure)) \quad \text{(A.22)}$$

$$StructureOfCompartmentOfAbdomen \sqsubseteq AbdominalStructure \sqcap StructureOfBodyCompartment \quad \text{(A.23)}$$

$$DisorderOfBodyWall \equiv DisorderByBodySite \sqcap (\exists findingSite.BodyWallStructure) \quad \text{(A.24)}$$

$$StructureOfImmuneSystem \sqsubseteq BodySystemStructure \quad \text{(A.25)}$$

$$JointStructureOfTrunk \sqsubseteq RegionalJointStructure \sqcap TrunkStructure \quad \text{(A.26)}$$

$$StructureOfHalfOfTrunkLateralToMidsagittalPlane \equiv StructureOfRightSideOfTrunk \tag{A.27}$$

$$\begin{aligned} InjuryOfFace \equiv\ & DisorderOfFace \sqcap InjuryOfHead \\ & \sqcap ((\exists associatedMorphology.TraumaticAbnormality) \\ & \sqcap (\exists findingSite.FaceStructure)) \end{aligned} \tag{A.28}$$

$$ChestAndAbdomen \sqsubseteq ChestAbdomenAndPelvis \sqcap NeckChestAndAbdomen \tag{A.29}$$

$$\begin{aligned} KidneyLesion \equiv\ & KidneyFinding \sqcap ((\exists associatedMorphology.Lesion) \\ & \sqcap (\exists findingSite.KidneyStructure)) \end{aligned} \tag{A.30}$$

$$\begin{aligned} BoneInjury \equiv\ & DisorderOfBone \sqcap InjuryOfMusculoskeletalSystem \\ & \sqcap ((\exists associatedMorphology.TraumaticAbnormality) \\ & \sqcap (\exists findingSite.BoneStructure)) \end{aligned} \tag{A.31}$$

$$\begin{aligned} CrystalArthropathyOfPelvis \equiv\ & CrystalArthropathy \\ & \sqcap ((\exists associatedMorphology.DepositionOfCrystallineMaterial) \\ & \sqcap (\exists findingSite.JointStructureOfPelvis)) \end{aligned} \tag{A.32}$$

$$\begin{aligned} FindingOfLowerLimb \equiv\ & FindingOfLimbStructure \\ & \sqcap (\exists findingSite.LowerLimbStructure) \end{aligned} \tag{A.33}$$

$$VulvaFinding \equiv FemaleGenitaliaFinding \sqcap (\exists findingSite.VulvalStructure) \tag{A.34}$$

$$StructureOfReticuloendothelialSystem \sqsubseteq StructureOfLymphoreticularSystem \tag{A.35}$$

$$InjuryOfLumbarSpine \equiv SpinalInjury \sqcap ((\exists associatedMorphology.TraumaticAbnormality) \sqcap (\exists findingSite.LumbarSpineStructure)) \quad (A.36)$$

$$TraumaticAndorNontraumaticInjuryOfAnatomicalSite \equiv TraumaticANDORNontraumaticInjury \sqcap ((\exists associatedMorphology.Damage) \sqcap (\exists findingSite.AnatomicalOrAcquiredBodyStructure)) \quad (A.37)$$

$$Hyperplasia \sqsubseteq GrowthAlteration \quad (A.38)$$

$$ProcedureOnFace \equiv ProcedureOnHead \sqcap (\exists procedureSite.FaceStructure) \quad (A.39)$$

$$LigamentInjury \equiv DisorderOfLigament \sqcap InjuryOfMusculoskeletalSystem \sqcap ((\exists associatedMorphology.TraumaticAbnormality) \sqcap (\exists findingSite.SkeletalLigamentStructure) \sqcap (\exists findingSite.StructureOfLigament)) \quad (A.40)$$

$$UpperBodyStructure \sqsubseteq BodyPartStructure \quad (A.41)$$

$$RemovalOfDeviceFromDigestiveSystem \equiv RemovalOfDevice \sqcap ((\exists directDevice.Device) \sqcap (\exists method.RemovalAction) \sqcap (\exists procedureSiteIndirect.StructureOfDigestiveSystem)) \quad (A.42)$$

$$CardiacPacemakerComponent \sqsubseteq CardiacImplant \sqcap CardiacPacemaker \quad (A.43)$$

$$FluoroscopyOfChest \equiv FluoroscopyOfRegions \sqcap ((\exists method.FluoroscopicImagingAction) \sqcap (\exists procedureSiteDirect.ThoracicStructure)) \quad \text{(A.44)}$$

$$PelvisAndLowerExtremities \sqsubseteq LowerBodyPartStructure \quad \text{(A.45)}$$

$$FluoroscopyOfRegions \equiv Fluoroscopy \sqcap ((\exists method.FluoroscopicImagingAction) \sqcap (\exists procedureSiteDirect.BodyRegionStructure)) \quad \text{(A.46)}$$

$$ProcedureOnBodySystem \equiv ProcedureBySite \sqcap (\exists procedureSite.BodySystemStructure) \quad \text{(A.47)}$$

$$HeartANDPericardiumStructure \sqsubseteq IntrathoracicCardiovascularStructure \sqcap MediastinalStructure \quad \text{(A.48)}$$

$$LooseBodyInJoint \equiv MechanicalJointDisorder \sqcap ((\exists associatedMorphology.LooseBody) \sqcap (\exists findingSite.JointStructure)) \quad \text{(A.49)}$$

$$ChestImaging \equiv ImagingProcedure \sqcap ((\exists method.ImagingAction) \sqcap (\exists procedureSiteDirect.ThoracicStructure)) \quad \text{(A.50)}$$

$$AbdominalStructure \sqsubseteq AbdomenAndPelvis \sqcap ChestAndAbdomen \sqcap LowerBodyPartStructure \sqcap StructureOfAbdominalSegmentOfTrunk \sqcap StructureOfSubregionOfTrunk \quad \text{(A.51)}$$

$$StructureOfHalfOfAbdomenLateralToMidsagittalPlane \equiv StructureOfRightHalfOfAbdomen \quad \text{(A.52)}$$

$$ProcedureByDevice \equiv Procedure \sqcap (\exists procedureDevice.Device) \quad \text{(A.53)}$$

$$MaleGenitalOrganStructure \sqsubseteq BodyOrganStructure \sqcap MaleGenitalStructure \sqcap StructureOfGenitalOrgan \quad \text{(A.54)}$$

$$StructureOfGenitourinarySystem \sqsubseteq BodySystemStructure \sqcap LowerTrunkStructure \sqcap PelvicStructure \quad \text{(A.55)}$$

$$FluoroscopyOfGastrointestinalTract \equiv Fluoroscopy \sqcap ((\exists method.FluoroscopicImagingAction) \sqcap (\exists procedureSiteDirect.GastrointestinalTractStructure)) \quad \text{(A.56)}$$

$$RadiographicProcedureOnCardiovascularSystem \equiv RadiographicImagingProcedure \sqcap ((\exists method.RadiographicImagingAction) \sqcap (\exists procedureSiteDirect.StructureOfCardiovascularSystem)) \quad \text{(A.57)}$$

$$StructureOfBodyCompartment \sqsubseteq BodySpaceStructure \sqcap StructureOfAnatomicalCompartment \sqcap TrunkStructure \quad \text{(A.58)}$$

$$StructureOfCompartmentOfThorax \sqsubseteq StructureOfBodyCompartment \sqcap ThoracicStructure \quad \text{(A.59)}$$

$$LargeIntestinalStructure \sqsubseteq IntestinalStructure \sqcap StructureOfAbdominalViscus \sqcap StructureOfViscusOfAbdomenProper \quad \text{(A.60)}$$

A.2 Questões de Competência para Checar os Axiomas Escolhidos para Testar o Verificador

Tabela A.1: Questões de competência geradas para checar todos os axiomas escolhidos randomicamente no teste do Verificador

Axioma	Questão de Competência	Resposta Esperada
A.1	Is allergen class a substance?	True
A.2	Is drug or medicament a substance?	True
A.3	Is amino acid supplement a protein supplementation?	True
A.4	Is synovial structure of limb a limb structure?	True
	Is synovial structure of limb a regional synovial structure?	True
A.5	Is congenital anomaly of body cavity a congenital anomaly of trunk?	True
	Is congenital anomaly of body cavity a disease?	True
	What is the associated morphology of congenital anomaly of body cavity?	Congenital anomaly
	Does congenital anomaly of body cavity have associated morphology some developmental abnormality?	True
	Where is the finding site of congenital anomaly of body cavity?	Body cavity structure
	What is the occurrence of congenital anomaly of body cavity?	Congenital
A.6	Is dysplasia a growth alteration?	True
	Is dysplasia a lesion?	True
A.7	Is procedure on skeletal system a procedure on musculoskeletal system?	True
	What is the procedure site of procedure on skeletal system?	skeletal system structure

A.8	Is abdomen destructive procedure a destructive procedure?	True
	Is abdomen destructive procedure a procedure on abdomen?	True
	What is the method of abdomen destructive procedure?	Destruction action
	Where is the procedure site direct of abdomen destructive procedure?	Abdominal structure
A.9	Is serous sac structure a body organ structure?	True
A.10	Is leg destructive procedure a destructive procedure?	True
	What is the method of leg destructive procedure?	Destruction action
	Where is the procedure site direct of leg destructive procedure?	Lower leg structure
A.11	Is finding of region of thorax a finding of trunk structure?	True
	What is the finding site of finding of region of thorax?	Thoracic structure
A.12	Is drug allergen or pseudoallergen an allergen or pseudoallergen?	True
A.13	Is abdominal cavity structure an abdominal structure?	True
	Is abdominal cavity structure a body cavity structure?	True
	Is abdominal cavity structure a structure of body cavity subdivision?	True
	Is abdominal cavity structure a structure of compartment of abdomen?	True
A.14	Is disorder of bone a disorder of skeletal system?	True
	What is the finding site of disorder of bone?	Bone structure
A.15	Is lower trunk structure a lower body part structure?	True
	Is lower trunk structure a structure of subregion of trunk?	True
A.16	Is right lower extremity structure a lower limb structure?	True

	Is right lower extremity structure a structure of right half of body?	True
A.17	Is structure of viscus a body organ structure?	True
A.18	Is destruction action a surgical action?	True
A.19	Is surgical procedure on thorax a procedure on thorax?	True
	Is surgical procedure on thorax a surgical procedure?	True
	Does surgical procedure on thorax have method some surgical action?	True
	Where is the procedure site of surgical procedure on thorax?	Thoracic structure
A.20	Is fixation a procedure by method?	True
	What is the method of fixation?	Fixation action
A.21	Is morphologically altered structure a body structure?	True
A.22	Is traumatic injury of external genitalia a genital injury?	True
	What is the associated morphology of traumatic injury of external genitalia?	Traumatic abnormality
	Where is the finding site of traumatic injury of external genitalia?	External genitalia structure
A.23	Is structure of compartment of abdomen a abdominal structure?	True
	Is structure of compartment of abdomen a body compartment?	True
A.24	Is disorder of body wall a disorder by body site?	True
	What is the finding site of disorder of body wall?	Body wall structure
A.25	Is structure of immune system a body system structure?	True
A.26	Is joint structure of trunk a regional joint structure?	True
	Is joint structure of trunk a trunk structure?	True

A.27	Is structure of half of trunk lateral to midsagittal plane a structure of right side of trunk?	True
A.28	Is injury of face a disorder of face?	True
	Is injury of face a injury of head?	True
	What is the associated morphology of injury of face?	Traumatic abnormality
	Where is the finding site of injury of face?	Face structure
A.29	Is chest and abdomen a chest abdomen and pelvis?	True
	Is chest and abdomen a neck chest and abdomen?	True
A.30	Is kidney lesion a kidney finding?	True
	Does kidney lesion have associated morphology some lesion?	True
	What is the finding site of kidney lesion?	Kidney structure
A.31	Is bone injury a disorder of bone?	True
	Is bone injury a injury of musculoskeletal system?	True
	What is the associated morphology of bone injury?	Traumatic abnormality
	Where is the finding site of bone injury?	Bone structure
A.32	Is crystal arthropathy of pelvis a crystal arthropathy?	True
	What is the associated morphology of crystal arthropathy of pelvis?	Deposition of crystalline material
	What is the finding site of crystal arthropathy of pelvis?	Joint structure of pelvis
A.33	Is finding of lower limb a finding of limb structure?	True
	Where is the finding site of finding of lower limb?	Lower limb structure
A.34	Is vulva finding a female genitalia finding?	True
	What is the finding site of vulva finding?	Vulval structure
A.35	Is structure of reticuloendothelial system a structure of lymphoreticular system	True

A.36	Is injury of lumbar spine a spinal injury?	True
	Does injury of lumbar spine have associated morphology some traumatica abnormality?	True
	Where is the finding site of injury of lumbar spine?	Lumbar spine structure
A.37	Is traumatic and or nontraumatic injury of anatomical site a traumatic and or bontraumatic injury?	True
	What is the associated morphology of traumatic and or nontraumatic injury of anatomical site?	Damage
	What is the finding site of traumatic and or nontraumatic injury of anatomical site?	Anatomical or acquired body structure
A.38	Is hyperplasia a growth alteration?	True
A.39	Is procedure on face a procedure on head?	True
	What is the procedure site of procedure on face?	Face structure
A.40	Is ligament injury a disorder of ligament?	True
	Is ligament injury a injury of musculoskeletal system?	True
	What is the associated morphology of ligament injury?	Traumatic abnormality
	What is the finding site of ligament injury?	Skeletal ligament structure
	Where is the finding site of ligament injury?	Structure of ligament
A.41	Is upper body structure a body part structure?	True
A.42	Is removal of device from digestive system a removal of device?	True
	What is the direct device of removal of device from digestive system?	Device
	Does removal of device from digestive system have method some removal action?	True
	Where is the procedure site indirect of removal of device from digestive system	Structure of digestive system
A.43	Is cardiac pacemaker component a cardiac implant?	True
	Is cardiac pacemaker component a cardiac pacemaker?	True
A.44	Is fluoroscopy of chest a fluoroscopy of regions?	True

	What is the method of fluoroscopy of chest?	Fluoroscopic imaging action
	Where is the procedure site direct of fluoroscopy of chest?	Thoracic structure
A.45	Is pelvis and lower extremities a lower body part structure?	True
A.46	Is fluoroscopy of regions a fluoroscopy?	True
	Does fluoroscopy of regions have method some fluoroscopic imaging action?	True
	What is the procedure site direct of fluoroscopy of regions?	Body region structure
A.47	Is procedure on body system a procedure by site?	True
	What is the procedure site of procedure on body system?	Body system structure
A.48	Is heart and pericardium structure a intrathoracic cardiovascular structure?	True
	Is heart and pericardium structure a intrathoracic mediastinal structure?	True
A.49	Is loose body in joint a mechanical joint disorder?	True
	What is the associated morphology of loose body in joint?	Loose body
	Where is the finding site of loose body in joint?	Joint structure
A.50	Is chest imaging a imaging procedure?	True
	What is the method of chest imaging?	Imaging action
	Where is the procedure site direct of chest imaging?	Thoracic structure
A.51	Is abdominal structure an abdomen and pelvis?	True
	Is abdominal structure a chest and abdomen?	True
	Is abdominal structure a lower body part structure?	True
	Is abdominal structure a structure of abdominal segment of trunk?	True
	Is abdominal structure a structure of structure of subregion of trunk?	True

A.52	Is structure of half of abdomen lateral to midsagittal plane a structure of right half of abdomen?	True
A.53	Is procedure by device a procedure?	True
	What is the procedure device of procedure by device?	Device
A.54	Is male genital organ structure a body organ structure?	True
	Is male genital organ structure a male genital structure?	True
	Is male genital organ structure a structure of genital organ?	True
A.55	Is structure of genitourinary system a body system structure?	True
	Is structure of genitourinary system a lower trunk structure?	True
	Is structure of genitourinary system a pelvic structure?	True
A.56	Is fluoroscopy of gastrointestinal tract a fluoroscopy?	True
	Does fluoroscopy of gastrointestinal tract have method some fluoroscopic imaging action?	True
	What is the procedure site direct of fluoroscopy of gastrointestinal tract?	Gastrointestinal tract structure
A.57	Is radiographic procedure on cardiovascular system a radiographic imaging procedure?	True
	What is the method of radiographic procedure on cardiovascular system?	Radiographic imaging action
	Where is the procedure site direct of radiographic procedure on cardiovascular system?	Structure of cardiovascular system
A.58	Is structure of body compartment a body space structure?	True
	Is structure of body compartment a structure of anatomical compartment?	True
	Is structure of body compartment a trunk structure?	True
A.59	Is structure of compartment of thorax a structure of body compartment?	True

	Is structure of compartment of thorax a thoracic structure?	True
A.60	Is large intestinal structure a intestinal structure?	True
	Is large intestinal structure a structure of abdominal viscus?	True
	Is large intestinal structure a structure of viscus of abdomen proper?	True

A.3 Axiomas Inferidos Escolhidos para Testar o Verificador

$$BloodMaterial \sqsubseteq HematologicAgent \quad \text{(A.61)}$$

$$MassOfBodyStructure \sqsubseteq FindingBySite \quad \text{(A.62)}$$

$$SurgicalProcedureOnLowerExtremity \sqsubseteq LimbOperation \quad \text{(A.63)}$$

$$AminoAcidSupplement \sqsubseteq ProteinSupplementation \quad \text{(A.64)}$$

$$ProcedureOnSoftTissue \sqsubseteq ProcedureBySite \quad \text{(A.65)}$$

$$BoneStructure \sqsubseteq BoneAndorJointStructure \quad \text{(A.66)}$$

$$StructureOfSubregionOfTrunk \sqsubseteq StructureOfLeftSideOfTrunk \quad \text{(A.67)}$$

$$MassOfSoftTissue \sqsubseteq GeneralFindingOfSoftTissue \quad \text{(A.68)}$$

$$PleuraFinding \sqsubseteq FindingOfRegionOfThorax \quad \text{(A.69)}$$

$$DestructiveProcedureOnLowerLimb \sqsubseteq DestructiveProcedureByAnatomicSite \quad \text{(A.70)}$$

$$KneeJointStructure \sqsubseteq StructureOfLeftKnee \quad \text{(A.71)}$$

$$StructureOfRightSideOfVulva \sqsubseteq LowerFemaleGenitalStructure \quad (A.72)$$

$$DisorderOfSkeletalSystem \sqsubseteq DiseaseOfMusculoskeletalSystem \quad (A.73)$$

$$SkinANDSubcutaneousTissueStructure \sqsubseteq StructureOfRightHalfOfBody \quad (A.74)$$

$$Lichen \sqsubseteq PapularEruption \quad (A.75)$$

$$KidneyStructure \sqsubseteq KidneyAndorUreterStructures \quad (A.76)$$

$$RightKidneyStructure \sqsubseteq StructureOfViscusOfAbdomenProper \quad (A.77)$$

$$DisorderOfAbdominalWall \sqsubseteq DisorderOfBodyWall \quad (A.78)$$

$$DiseaseOfLymphNode \sqsubseteq DisorderOfLymphoidSystem \quad (A.79)$$

$$BoneStructure \sqsubseteq SkeletalSystemStructure \quad (A.80)$$

$$CardiovascularStructureOfTrunk \sqsubseteq TrunkStructure \quad (A.81)$$

$$ProcedureByDevice \sqsubseteq Procedure \quad (A.82)$$

$$StructureOfBodyCavitySubdivision \sqsubseteq BodyCavityStructure \quad (A.83)$$

$$FluoroscopyOfSystems \sqsubseteq ProcedureOnBodySystem \quad (A.84)$$

$$StructureOfRightSideOfTrunk \sqsubseteq BodyPartStructure \quad (A.85)$$

$$StructureOfLeftHalfOfAbdomen \sqsubseteq StructureOfAbdominalSegmentOfTrunk \quad (A.86)$$

$$StructureOfLeftTestis \equiv StructureOfRightTestis \equiv TestisStructure \equiv UnilateralTestis \tag{A.87}$$

$$DisorderOfPelvicRegionOfTrunk \sqsubseteq DisorderOfTrunk \tag{A.88}$$

$$SpecimenFromTestis \sqsubseteq GonadSample \tag{A.89}$$

$$DiseaseOfMediastinum \sqsubseteq MediastinalFinding \tag{A.90}$$

A.4 Axiomas de Equivalência Retirados de Módulos para Testar o Questionador

$$DiseaseOfMusculoskeletalSystem \equiv DisorderByBodySite \sqcap (\exists findingSite.StructureOfMusculoskeletalSystem) \tag{A.91}$$

$$UrogenitalFinding \equiv FindingBySite \sqcap (\exists findingSite.StructureOfGenitourinarySystem) \tag{A.92}$$

$$AllergicCondition \equiv Disease \sqcap HypersensitivityCondition \sqcap (\exists pathologicalProcess.AllergicProcess) \tag{A.93}$$

$$OEGenitalia \equiv GenitalFinding \sqcap OESpecifiedExaminationFindings \sqcap (\exists findingInformer.PerformerOfMethod) \sqcap (\exists findingMethod.PhysicalExaminationProcedure) \sqcap (\exists findingSite.GenitalStructure) \tag{A.94}$$

$$MalignantNeoplasmOfGenitourinaryOrgan \equiv DiseaseOfTheGenitourinarySystem \sqcap ((\exists associatedMorphology.MalignantNeoplasmMorphology) \sqcap (\exists findingSite.StructureOfGenitourinarySystem)) \tag{A.95}$$

$$CongenitalFemaleUrogenitalAnomaly \equiv Disease \sqcap GenitourinaryCongenitalAnomalies \sqcap ((\exists associatedMorphology.CongenitalAnomaly) \sqcap (\exists associatedMorphology.DevelopmentalAbnormality) \sqcap (\exists findingSite.FemaleGenitourinarySystemStructure) \sqcap (\exists occurrence.Congenital)) \tag{A.96}$$

$$NeoplasmOfFace \equiv DisorderOfFace \sqcap NeoplasmOfHead \sqcap ((\exists associatedMorphology.Neoplasm) \sqcap (\exists findingSite.FaceStructure)) \tag{A.97}$$

$$ProcedureOnSkeletalSystem \equiv ProcedureOnMusculoskeletalSystem \sqcap (\exists procedureSite.SkeletalSystemStructure) \tag{A.98}$$

$$FemaleGenitaliaFinding \equiv FemaleReproductiveFinding \sqcap GenitalFinding \sqcap (\exists findingSite.FemaleGenitalStructure) \tag{A.99}$$

$$\begin{aligned} CongenitalAnomalyOfFace \equiv\ & CongenitalAnomalyOfHead \\ & \sqcap Disease \\ & \sqcap ((\exists associatedMorphology.CongenitalAnomaly) \\ & \sqcap (\exists associatedMorphology.DevelopmentalAbnormality) \\ & \sqcap (\exists findingSite.FaceStructure) \\ & \sqcap (\exists occurrence.Congenital)) \\ & \sqcap (\exists occurrence.Congenital) \end{aligned} \tag{A.100}$$

$$\begin{aligned} BodyFluidAnalysis \equiv\ & LaboratoryTest \\ & \sqcap (\exists hasSpecimen.BodyFluidSample) \\ & \sqcap (\exists method.EvaluationAction) \end{aligned} \tag{A.101}$$

$$\begin{aligned} ExcisionOfThoracicVaricoseVein \equiv\ & ExcisionOfVein \\ & \sqcap ((\exists directMorphology.Varix) \\ & \sqcap (\exists method.ExcisionAction) \\ & \sqcap (\exists procedureSiteDirect.StructureOfVeinOfThorax)) \end{aligned} \tag{A.102}$$

$$\begin{aligned} ProcedureOnDigestiveTract \equiv\ & ProcedureOnDigestiveSystem \\ & \sqcap (\exists procedureSite.DigestiveTractStructure) \end{aligned} \tag{A.103}$$

$$\begin{aligned} NeoplasmOfGastrointestinalTract \equiv\ & NeoplasmOfDigestiveTract \\ & \sqcap ((\exists associatedMorphology.Neoplasm) \\ & \sqcap (\exists findingSite.GastrointestinalTractStructure)) \end{aligned} \tag{A.104}$$

$$\begin{aligned} FindingOfSpinalRegion \equiv\ & FindingBySite \\ & \sqcap (\exists findingSite.StructureOfVertebralRegionOfBack) \end{aligned} \tag{A.105}$$

$$FistulaOfUterus \equiv DisorderOfUterus \sqcap FistulaOfTheFemaleGenitalOrgans \sqcap ((\exists associatedMorphology.Fistula) \sqcap (\exists findingSite.UterineStructure)) \quad \text{(A.106)}$$

$$ProcedureOnNeoplasmOfEyeRegion \equiv ProcedureOnEyeRegion \sqcap ((\exists procedureMorphology.Neoplasm) \sqcap (\exists procedureSite.EyeRegionStructure)) \quad \text{(A.107)}$$

$$DisorderOfAbdomen \equiv DisorderOfTrunk \sqcap (\exists findingSite.AbdominalStructure) \quad \text{(A.108)}$$

$$NeoplasmOfIntraabdominalOrgans \equiv NeoplasmOfAbdomen \sqcap ((\exists associatedMorphology.Neoplasm) \sqcap (\exists findingSite.StructureOfAbdominalViscus)) \quad \text{(A.109)}$$

$$StructureOfHalfOfHeadLateralToMidsagittalPlane \equiv HeadPart \sqcap StructureOfHalfOfBodyLateralToMidsagittalPlane \quad \text{(A.110)}$$

$$RespiratoryTractHemorrhage \equiv DiseaseOfRespiratorySystem \sqcap ((\exists associatedMorphology.Hemorrhage) \sqcap (\exists findingSite.RespiratoryTractStructure)) \quad \text{(A.111)}$$

$$Reimplantation \equiv Implantation \sqcap (\exists method.ReimplantationAction) \quad \text{(A.112)}$$

$$DisorderOfLeftCardiacVentricle \equiv DisorderOfCardiacVentricle \sqcap (\exists findingSite.LeftVentricularStructure) \quad \text{(A.113)}$$

$$ProcedureOnStomach \equiv ProcedureOnDigestiveOrgan \sqcap (\exists procedureSite.StomachStructure) \quad \text{(A.114)}$$

$$\begin{aligned} CongenitalAnomalyOfCardiovascularStructureOfTrunk \equiv\ & CongenitalAnomalyOfTrunk \\ & \sqcap Disease \\ & \sqcap ((\exists associatedMorphology.CongenitalAnomaly) \\ & \sqcap (\exists associatedMorphology.DevelopmentalAbnormality) \\ & \sqcap (\exists findingSite.CardiovascularStructureOfTrunk) \\ & \sqcap (\exists occurrence.Congenital)) \\ & \sqcap (\exists occurrence.Congenital) \end{aligned} \quad \text{(A.115)}$$

$$\begin{aligned} OperativeProcedureOnLowerLeg \equiv\ & SurgicalProcedureOnLowerExtremity \\ & \sqcap ((\exists method.SurgicalAction) \\ & \sqcap (\exists procedureSite.LowerLegStructure)) \end{aligned} \quad \text{(A.116)}$$

$$ProcedureOnBloodVessel \equiv ProcedureOnCardiovascularSystem \sqcap (\exists procedureSite.BloodVesselStructure) \quad \text{(A.117)}$$

$$\begin{aligned} ChestInjury \equiv\ & InjuryOfTrunk \\ & \sqcap ((\exists associatedMorphology.TraumaticAbnormality) \\ & \sqcap (\exists findingSite.ThoracicStructure)) \end{aligned} \quad \text{(A.118)}$$

$$\begin{aligned} EvaluationOfMusculoskelatalSystem \equiv\ & EvaluationProcedure \\ & \sqcap ((\exists method.EvaluationAction) \\ & \sqcap (\exists procedureSiteDirect.StructureOfMusculoskeletalSystem)) \end{aligned} \quad \text{(A.119)}$$

$$OperationOnPapillaryMuscleOfHeart \equiv OperationOnHeart \sqcap ((\exists method.SurgicalAction) \sqcap (\exists procedureSite.StructureOfPapillaryMuscle)) \quad (A.120)$$

A.5 Listas de Tarefas dos Estudos de Caso

A.5.1 Estudo de Caso com Pesquisadores

1. Checar a QC: “Is driver an adult?”, com a resposta: “True”;
2. Checar a QC: “What is the employer of truck driver?”, com a resposta: “Truck company”;
3. Efetuar as tarefas usando o Rastreador:
   (a) Verificar quais QCs foram checadas;
   (b) Verificar o axioma correspondente a cada QC;
   (c) Analisar os conceitos que foram impactados por cada QC;
4. Checar a QC: “Is pilot a driver?”, com a resposta: “True”;
5. Utilizar o Questionador para escolher e responder uma QS;
6. Checar a QC: “Does professor have employer some university?”, com a resposta: “True”;
7. Utilizar o Questionador para escolher e responder uma QS;
8. Checar a QC: “is bank clerk a bank employee?”, com a resposta: “True”;
9. Utilizar o Questionador para escolher e responder uma QS;
10. Efetuar as tarefas usando o Rastreador:
    (a) Verificar quais QCs foram checadas;
    (b) Verificar o axioma correspondente a cada QC;
    (c) Analisar os conceitos que foram impactados por cada QC;
    (d) Analisar o código alterado pelo conhecimento adicionado pela QS.

A.5.2 Estudo de Caso com Alunos da Disciplina Introdução à Informática em Saúde

1. Checar a QC: “Is arthropathy a disease of musculoskeletal system?”, com a resposta: “True”;
2. Checar a QC: “What is the finding site of infection of pelvis?”, com a resposta: “Bone structure of pelvis”;
3. Efetuar as tarefas usando o Rastreador:
    (a) Verificar quais QCs foram checadas;
    (b) Verificar o axioma correspondente a cada QC;
    (c) Analisar os conceitos que foram impactados por cada QC;
4. Checar a QC: “Is disease of musculoskeletal system a musculoskeletal finding?”, com a resposta: “True”;
5. Utilizar o Questionador para escolher e responder uma QS;
6. Checar a QC: “Is urogenital finding a finding of pelvic structure?”, com a resposta: “True”;
7. Utilizar o Questionador para escolher e responder uma QS;
8. Checar a QC: “Does allergic condition have pathological process some hypersensitivity process?”, com a resposta: “True”;
9. Utilizar o Questionador para escolher e responder uma QS;
10. Efetuar as tarefas usando o Rastreador:
    (a) Verificar quais QCs foram checadas;
    (b) Verificar o axioma correspondente a cada QC;
    (c) Analisar os conceitos que foram impactados por cada QC;
    (d) Analisar o código alterado pelo conhecimento adicionado pela QS.