Pós-Graduação em Ciência da Computação

Camila Bezerra da Silva

UMA ABORDAGEM DE MODULARIZAÇÃO DE ONTOLOGIAS BASEADA NA SATISFAÇÃO LÓGICA DE QUESTÕES DE COMPETÊNCIA

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

Recife
2019

Camila Bezerra da Silva

UMA ABORDAGEM DE MODULARIZAÇÃO DE ONTOLOGIAS BASEADA NA SATISFAÇÃO LÓGICA DE QUESTÕES DE COMPETÊNCIA

Tese de Doutorado apresentada ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de Doutor em Ciência da Computação.

Área de Concentração: Inteligência Computacional
Orientador: Frederico Luiz Gonçalves de Freitas

Recife
2019

Catalogação na fonte

Bibliotecário Vimário Carvalho CRB4-1204

S586s Silva, Camila Bezerra da.
Uma abordagem de modularização de ontologias baseada na satisfação lógica de questões de competências / Camila Bezerra da Silva. – 2019.
146 f.: il., fig., tab.

Orientador: Frederico Luiz Gonçalves de Freitas.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2019.
Inclui referências e apêndice.

1. Inteligência computacional. 2. Modularização de ontologias. 3. Questões de competências. I. Freitas, Frederico Luiz (orientador). II. Título.

006.3 CDD 22.ed. UFPE-MEI 2019-134

Camila Bezerra da Silva

" UMA ABORDAGEM DE MODULARIZAÇÃO DE ONTOLOGIAS BASEADA NA SATISFAÇÃO LÓGICA DE QUESTÕES DE COMPETÊNCIA"

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 Doutor em Ciência da Computação.

Aprovado em: 07/03/2019.

Orientador: Prof. Dr. **Frederico Luiz Gonçalves de Freitas**

BANCA EXAMINADORA

Profa. Dra. Anjolina Grisi de Oliveira
Centro de Informática / UFPE

Prof. Dr. Leopoldo Motta Texeira
Centro de Informática / UFPE

Prof. Dr. José Maria Parente de Oliveira
Divisão de Ciência da Computação / ITA

Profa. Dra. Renata Wasserman
Instituto de Matemática e Estastíca / USP

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

Eu dedico essa tese a minha avó, Josefa Bezerra (in memoriam), mulher simples, praticamente analfabeta, mas que compreendia a importância do estudo, da busca pelo conhecimento. E se hoje cheguei aonde eu estou, é graças a ela e a Deus.

AGRADECIMENTOS

Agradeço ao meu orientador, Fred Freitas, por todo suporte, paciência e orientação. Fred, não foi apenas um orientador do trabalho de tese, mas um amigo, que sempre procurou auxiliar no desenvolvimento da tese, como também sabia identificar quando seus orientandos não estavam bem, e sempre trazia conforto com seus conselhos e atenção.

Agradeço a minha mãe, Jacira Bezerra, que sempre esteve do meu lado, apoiando minhas escolhas, sofrendo e se alegrando com cada derrota e vitória. Que mesmo com a distância, sempre tive uma conversa amorosa, que acalmava meu coração, todas as noites. Soube com paciência e sabedoria me orientar, me dá forças em todo esse período.

Agradeço a meu namorado, Lúcio, que mesmo a distância, com todos os problemas que surgiram nessa jornada, esteve presente, me apoiou, foi paciente, compreensivo e amoroso.

Agradeço a meus amigos mais íntimos, que estiveram presente, me escutaram, me alegraram, vivenciaram comigo essa experiência.

E por fim, agradeço a Deus, pela graça concedida de fazer essa tese, toda ajuda divida que recebi nos momentos de provação, pelo aconchego recebido pela minha mãe Maria Santissíma, quando mais precisei.

RESUMO

Existem várias abordagens propostas para modularização, porém a questão de inconsistência lógica entre módulos é pouco explorada. De fato, a inconsistência pode ser resultado de questões de competência inconsistentes, algo nunca explorado na literatura. As QCs são fundamentais para o desenvolvimento de ontologias, já que representam os requisitos de uma ontologia. A proposta deste trabalho é justamente explorar essa lacuna através de uma abordagem para implementação de modularização de ontologias e checagem de consistência utilizando questões de competência. A vantagem principal desse tipo de abordagem é justamente prover checagem de consistência entre QCs, mesmo que estejam em módulos diferentes. Isso é possível quando partimos da fase inicial do desenvolvimento de ontologias, ou seja, quando só existem as QCs e ainda não há a ontologia. No caso, as QCs são modularizadas e a consistência, entre elas, é verificada. Este trabalho provê principalmente as seguintes contribuições: (1) Permitir um melhor entendimento das QCs, especialmente em ontologias muito grandes, e o reuso, por meio da modularização; e (2) Checar inconsistências entre QCs durante o início do desenvolvimento, consequentemente, entre módulos, poupando tempo e trabalho nas fases posteriores.

Palavras-chaves: Modularização de ontologias. Questões de competência. Satisfatibilidade.

ABSTRACT

There are several approaches proposed for modularization, but the question of inconsistency between modules is little explored. In fact inconsistencies may be due to inconsistency of competence questions, something that has never been explored in the literature. The Competency Questions(CQs) are fundamental to the development of ontologies, as they represent the requirements of an ontology. The purpose of this thesis is to exploit that gap, by developing an approach and implementation of ontology modularization using competency questions. The main advantage of this approach is precisely to provide consistency checking among competency questions, even if they are in different modules. The approach applies in situations in which starting from the scratch, i.e., there are only the competency questions, and there is no ontology yet. In this case competency questions are modularized and consistency among them checked. This work mainly provides the following contributions: (1) To allow a better understanding of the competency questions, especially in large ontologies, and the reuse, by modularization; and (2) Check inconsistencies among CQs during the development, saving time and effort in the later stages.

Keywords: Ontology Modularization. Competency Question. Satisfatibility.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABox	Assertional Axioms
FOL	First Order Logic
LD	Lógica Descritiva
OWL	Ontology Web Language
PLN	Processamento de Linguagem Natural
QCs	Questões de Competência
RDF	Resource Description Framework
SPARQL	Semantic Query Language for Databases
TBox	Terminological Axioms
URI	Uniform Resource Indicator
W3C	World Wide Web Consortium
XML	eXtensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTO

A partir da popularização do uso de ontologias e, em particular, após o surgimento de repositórios, como a Ontolingua ((FARQUHAR; FIKES; RICE, 1997)), dentre outros e com o advento da *Web Semântica* ((BERNERS-LEE; HENDLER; LASSILA, 2001)), houve um crescimento considerável de pesquisas na área de ontologias, como o surgimento de ferramentas e metodologias para apoiar o desenvolvimento delas.

Ontologias podem ser consideradas como uma forma de representar um modelo de um domínio específico. De acordo com (USCHOLD; G RUNINGER, 1 996) as ontologias podem solucionar problemas relacionados à comunicação entre pessoas e sistemas de software. Além disso, ajudam a solucionar ou eliminar problemas com relação à falta de conhecimento sobre conceitos envolvidos nos processos de comunicação.

1.1.1 Modularização de Ontologias

Existem várias abordagens e ferramentas propostas para reutilizar ontologias. A maioria das metodologias também contempla, em uma das fases do desenvolvimento, o reuso. No entanto, parte dessas ferramentas propostas foram descontinuadas. Atualmente, as ferramentas disponibilizadas e mais atuais, como a Hys ((MARTIN-RECUERDA; WALTHER, 2015)) e a Manchester API ((HORRIDGE; BECHHOFER, 2011)) possuem uma função para extração de módulos de ontologias, visando o reuso do conhecimento extraído.

A noção de modularização na Engenharia de Software se refere ao desenvolvimento de software de uma forma estruturada, que suporta a combinação de componentes independentes, fáceis de construir, reutilizar e manter ((PARNAS, 1972)). Segundo (D'AQUIN et al., 2007), do ponto de vista da Engenharia de Ontologias,

> "modularização deve ser considerada como uma forma de estruturar ontologias. Isso significa que a construção de uma ontologia grande deveria ser baseada na combinação de componentes de conhecimento autocontidos, independentes e reutilizáveis."

1.1.2 Questões de Competência

Por outro lado, uma etapa que faz parte da maioria das metodologias de ontologias é a de especificação de requisitos da ontologia, ou seja, a criação de questões de competência especificadas em linguagem natural cuja ontologia resultante após o seu desenvolvimento deve ser capaz de respondê-las ((NOY; HAFNER, 1997)). Por exemplo, supondo que a definição de uma pizza marguerita seja "uma pizza com queijo, tomate e manjericão",

em relação à ontologia de pizza[1], uma QC pode ser formada pela pergunta "A Pizza Marguerita é uma pizza vegetariana?" junto com a resposta "Sim". Assim, a QC só será considerada satisfeita se a ontologia sobre Pizzas conseguir respondê-la corretamente, ou seja, se existe um axioma que afirme que pizzas vegetarianas não contém carne, nem peixe, nem frango e um conjunto de axiomas ajude a deduzir automaticamente a resposta por meio de um raciocinador automático.

Questões de Competência (QCs) foram primeiramente introduzidas na literatura por (FOX; GRÜNINGER, 1994) em um trabalho que tratava sobre microteorias, que eram as partes das ontologias que deveriam conter um conjunto de axiomas necessário e suficiente para representar e resolver essas questões. Portanto, QCs foram definidas já ligadas à possibilidade de verificar, por meio de lógica e raciocínio automático, se poderiam ser satisfeitas por meio de inferências lógicas. Por isso, a partir dessa perspectiva, começamos a trabalhar com a ideia de gerar módulos utilizando QCs.

A maioria das metodologias sugere o uso de questões de competência para especificação da ontologia. No entanto, as metodologias descrevem superficialmente como utilizar as questões de competência. Na área de Engenharia de Requisitos de Software, (BOHM, 1987) afirma que

> "encontrar e consertar um problema de software após a entrega do software é cem vezes mais custoso do que encontrar e consertar o problema nas etapas iniciais do desenvolvimento".

Isso significa que contradições, negligências e ambiguidades não descobertas no início do desenvolvimento, frequentemente, resultam em consequências indesejadas. O mesmo problema pode existir na Engenharia de Ontologias, se as ontologias forem mal especificadas.

1.2 MOTIVAÇÃO E JUSTIFICATIVA

Como foi visto, a especificação de questões de competência pode ser mais explorada na Engenharia de Ontologias. Além do uso para especificação de ontologias, QCs podem ser empregadas para verificar se a ontologia satisfaz os seus requisitos, ou seja, na etapa de verificação e, por exemplo, na rastreabilidade de requisitos e modularização de ontologias. Existem alguns trabalhos sobre o uso de QCs na etapa de verificação de ontologias. Entretanto, até onde sabemos, a maioria apregoa a checagem de QCs manualmente e, no caso de trabalhos que utilizam verificação semiautomática, a verificação é feita somente no âmbito assercional da ontologia, ou seja, sobre instâncias ((ANGLES; GUTIERREZ, 2008)), e sem a possibilidade de raciocínio automático baseado em lógica.

[1] https://www.w3.org/TR/owl-guide/pizza.rdf

Logo, ainda existem muitas lacunas que podem ser exploradas na utilização de QCs para apoiar o ciclo todo de desenvolvimento de ontologias. Uma dessas lacunas é a modularização de ontologias utilizando QCs e a checagem de satisfatibilidade lógica entre QCs, objetivo que este trabalho se propõe a resolver. Na abordagem proposta, um módulo é considerado exatamente uma QC e tudo o que ela envolver. Tal proposta será explicada melhor no final deste capítulo.

A checagem de satisfatibilidade entre QCs traz um benefício muito importante que é checar inconsistências lógicas na fase inicial de especificações de requisitos, antes mesmo de construir a ontologia, poupando tempo e esforço, de forma semiautomática. Além disso, vale ressaltar que a inconsistência entre módulos é pouco explorada na literatura, e que tal inconsistência pode ser resultado de questões de competência inconsistentes, algo nunca explorado até então. Já a modularização no início do ciclo de desenvolvimento, ou seja, na fase de especificação de QCs, traz benefícios, como:

- Modularizar os requisitos, ou seja, as QCs, no início do desenvolvimento da ontologia, é útil principalmente em grandes ontologias, nas quais podem existir centenas de QCs;

- Modularizar uma QC, que neste trabalho iremos considerar como módulo, e a partir dela, criar módulos maiores com a junção de outras. Com as QCs modularizadas, o desenvolvedor terá uma visão mais clara dos requisitos, de como eles estão relacionados, e auxiliar nas fases posteriores, como a de manutenção.

1.3 PROBLEMA DE PESQUISA

A modularização na Engenharia de Software tem como objetivo dividir um software em pedaços menores que sejam independentes e reusáveis, no intuito de melhorar a compreensão do software e principalmente o reuso. Já na Engenharia de Ontologias, existem várias abordagens de modularização que não se aproveitam do fato de que a ontologia é um artefato lógico e os requisitos são perguntas e respostas. Portanto, a ideia da abordagem é criar os módulos a partir das questões de competência. Com isso ganham-se vários benefícios. Como explicado, uma questão de competência nada mais é do que um requisito de uma ontologia. Entretanto, a literatura não explora o fato de que pode haver QCs conflitantes. Por exemplo, de acordo com a ontologia de Pizza, um conjunto de QCs conflitantes poderia ser:
QC1 = Quais ingredientes tem uma pizza de bacalhau? Resposta = Queijo, bacalhau, azeitona, tomate, cebola;
QC2 = O que é uma pizza interessante? Resposta = Uma pizza que possui no máximo 3 sabores;
QC3 = A pizza de bacalhau é interessante? Resposta = sim

No exemplo acima, a pizza de bacalhau é definida com cinco sabores e a o mesmo tempo, é dito que ela é uma pizza interessante. No entanto, uma pizza interessante tem, no máximo, três sabores, o que provoca um conflito lógico, omo foi definido anteriormente.

Na abordagem proposta, durante a fase de especificação, quando um desenvolvedor define uma questão de competência em linguagem natural, a QC representará um módulo, com o vocabulário e os axiomas necessários para respondê-la corretamente.

Nesta tese, tentaremos identificar esses conflitos, definindo módulo de ontologia como uma questão de competência. Assim, quando houver conflitos entre os módulos, isso se deverá a conflitos entre as questões de competência. Esta tese tenta responder às seguintes perguntas: Quais os benefícios que as questões de competências podem trazer para a modularização de ontologias? De que forma as inconsistências entre QCs podem trazer problemas para a modularização de ontologias?

Para respondê-las, surgem as questões abaixo:

- É possível detectar inconsistências entre QCs através de módulos?
- A abordagem ajuda o desenvolvedor de ontologias a construir uma ontologia modular e consistente?
- Uma ferramenta que tanto modularize QCs, como verifique se há inconsistências entre QCs, ajuda os engenheiros de ontologias, poupando tempo e trabalho?

1.4 OBJETIVOS

O objetivo geral é criar uma abordagem para modularização de ontologias baseada em questões de competência, além da verificação de inconsistência lógica entre QCs e uma ferramenta que dê apoio à abordagem. Os objetivos específicos são:

1. Construir uma ferramenta que verifique se uma ontologia ou módulo satisfaz os requisitos, ou seja, as QCs;
2. Propor uma abordagem para modularização de ontologias baseada em QCs e criar um método de verificação de inconsistência lógica entre duas ou mais QCs (que são transformadas para Lógica de Descrições);
3. Implementar a abordagem proposta numa ferramenta com o propósito principal de validação.

1.5 VISÃO GERAL DO TRABALHO PROPOSTO

Para responder a uma questão de competência, talvez seja necessário responder a outras questões relacionadas à questão principal. Para a pergunta “Quais as melhores músicas clássicas do século IX?”, precisamos responder primeiro “o que é música?”, “O que é clássica?”, “O que é século”, por exemplo.

Na abordagem, essas questões derivadas de outras questões de competência são denominadas subQCs. Portanto, consideramos que uma QC possui subQCs. Além disso, uma

QC e suas subQCs constituem um módulo. A Figura 1 exibe uma QC modularizada, na qual para responder à QC3, é preciso responder à QC3.1 e à QC 3.1.1, e à QC3.1.1.1.

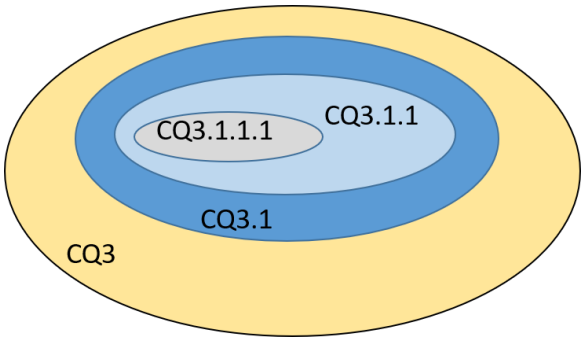


Figura 1 – Uma QC modularizada

No processo de modularização de uma ontologia, como cada QC constitui um módulo, pode acontecer de uma QC ou subQC existir em outro módulo. Nesse caso, os módulos teriam uma ligação, pois compartilham a mesma QC.

QCs são escritas em linguagem natural e precisam ser compreendidas pelo computador para que a abordagem seja automática ou semiautomática. As QCs, então, são traduzidas para uma linguagem lógica, na qual um raciocínio automático possa ser efetuado para responder às QCs corretamente.

O fluxo da abordagem é apresentado na Figura 2.

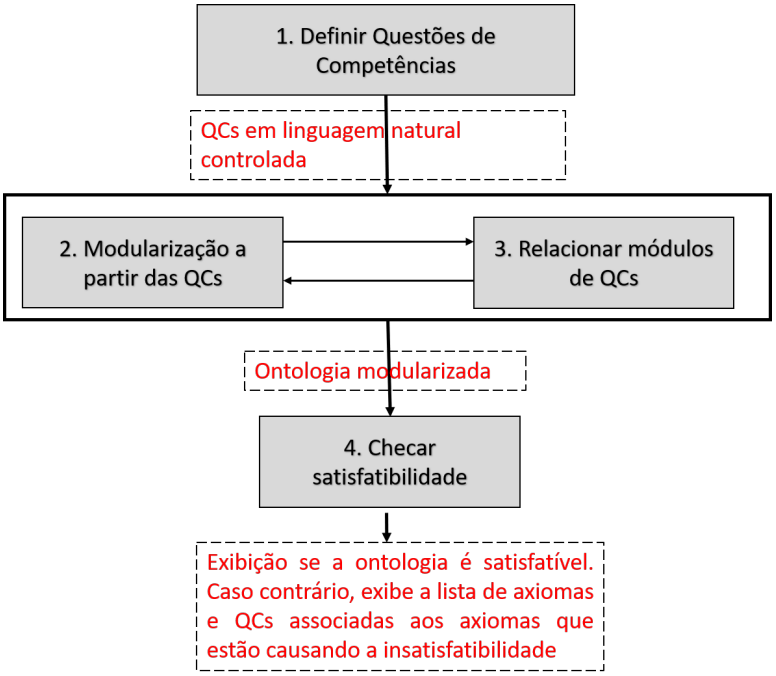


Figura 2 – Abordagem

1. O primeiro passo é criar questões de competências em linguagem natural controlada (a ser explicado na seção 6.4.4).

2. Depois, é realizada a modularização a partir de QCs, considerando que uma questão de competência pode gerar novas questões para poder respondê-la, que são chamadas subQCs. A QC e suas subQCs são consideradas um módulo, que pode ter conexões com outros módulos;

3. É verificado também se existe alguma QC já inserida. Se existir, ela será conectada e, consequentemente, os módulos que as representam;

4. Por fim, cada módulo e o conjunto de módulos são checados com relação à satisfatibilidade, descobrindo, dessa forma, ainda na fase inicial, erros de especificação.

O trabalho propôs uma abordagem de modularização de ontologias e a checagem de satisfatibilidade utilizando questões de competência. Essa abordagem traz benefícios inovadores para o desenvolvimento de ontologias:

- Modularizar uma ontologia já na fase inicial de desenvolvimento, a partir da especificação de requisitos da ontologia;
- O desenvolvedor terá conhecimento de que existem insatisfatibilidades durante a criação da ontologia, e não no final do desenvolvimento, como fazem as outras abordagens de modularização;
- Proporcionar reuso por meio de técnicas de modularização;
- Poupar tempo e trabalho, já que as insatisfatibilidades serão descobertas no início do desenvolvimento.

1.5.1 Experimentos

Foi criada uma ferramenta que implementa a abordagem (ver capítulo 7). Foram realizados experimentos com o desenvolvedor da ferramenta e com desenvolvedores externos. O experimento consistiu basicamente em:

- Criação de um conjunto de QCs para o desenvolvimento de ontologias já conhecidas, como as ontologias de Pizza, Wine e MadCows;
- Inserção dessas QCs na ferramenta;
- Inserção das respostas das QCs geradas;
- Após isso, foi verificado se a ontologia modular gerada pela ferramenta continha um vocabulário e axiomas similares com as ontologias já existentes (Pizza, Wine e MadCows);

- E checagem da existência de insatisfatibilidades. No caso da ontologia de Pizza e MadCow, já sabemos que em ambas existe uma insatisfatibilidade. Então, foi verificado se a ontologia gerada também continha essas insatisfatibilidades conhecidas;
- Foram aplicadas métricas para descobertas de padrões, coesão e acoplamento nos módulos;
- E foi aplicado um questionário qualitativo aos desenvolvedores externos sobre a experiência deles com a ferramenta.

1.6 ORGANIZAÇÃO DO TEXTO

Este trabalho está organizado em 9 capítulos descritos a seguir:

- Capitulo 2: Descreve ontologias e web semântica, que também é base deste trabalho;
- Capitulo 3: Descreve Lógica de Descrições, que é uma das bases deste trabalho;
- Capitulo 4: Apresenta o estado da arte sobre modularização de ontologias destacando os principais trabalhos;
- Capitulo 5: Apresenta o estado da arte sobre Questões de Competência, em que são destacados os principais trabalhos que utilizam QCs;
- Capitulo 6: Apresenta a proposta deste trabalho;
- Capitulo 7: Ferramenta CQSetConsistencyChecker e Experimentos;
- Capitulo 8: Ferramenta CQChecker;
- Capitulo 9: Considerações Finais e Trabalhos Futuros.

2 ONTOLOGIAS

A palavra ontologia é originada da Filosofia, tendo como objetivo principal o estudo da "natureza do ser". Segundo (CAMPBELL, 1981), visa caracterizar a realidade e os seres, por meio de identificação de categorias essenciais e das relações entre si. Na Filosofia, o primeiro autor que mencionou o termo ontologia foi um filósofo alemão chamado Jacob Lorhard (1561-1609), em 1606. No entanto, a popularização do termo surgiu depois da publicação do artigo denominado "Philosophia Prima sive Ontologia" por Christian Wolff no século XVIII (CORAZZON, 2016). Atualmente, do ponto de vista filosófico, a ontologia é uma base de conhecimento que contém um consenso certificado sobre itens desta base.

Na computação, a ontologia serve como um meio de especificação formal de um domínio, representando os conceitos, relações, axiomas, restrições e instâncias contidos nele. Além disso, provê um meio de especificação explícita de dados semânticos.

Os autores (SMITH; WELTY, 2001), utilizaram a definição "ontologias filosóficas"

> "Para denominar a ciência do que é, dos tipos e estruturas de objetos, propriedades, eventos, processos e relações em todas as áreas da realidade, interessada com a classificação definitiva e exaustiva das entidades em todas as esferas do ser."

2.1 COMPONENTES DE UMA ONTOLOGIA

Conforme (PEREZ; CORCHO, 2002), normalmente uma ontologia é composta por cinco componentes:

- **Classe ou conceito** - representa um conjunto de indivíduos com características comuns pertencentes a um domínio. Classes são normalmente organizadas em hierarquias, podendo ser classificadas em abstratas ou concretas, simples ou complexas, reais ou fictícias. A Figura 3 mostra duas classes: Person e Country.

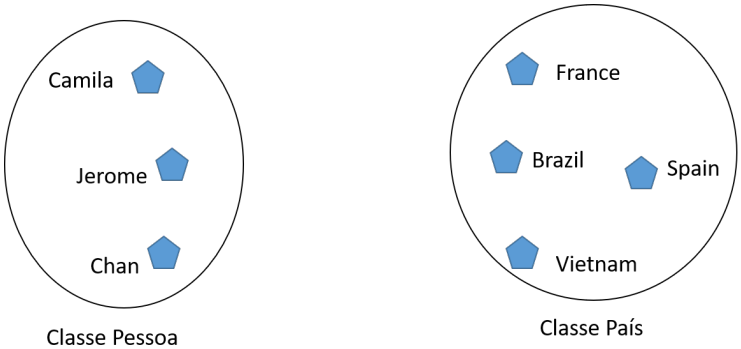


Figura 3 – Exemplo de Classe

- **Indivíduo ou instância** - representa elementos do domínio associados a um conceito específico. Por exemplo, “Camila” é um indivíduo da classe “Person” na Figura 3.

- **Relação** - representa conexões entre indivíduos de classes diferentes. A Figura 4 mostra que os indivíduos da classe “Person” estão relacionados com os indivíduos da classe “Country” por meio da propriedade “wasBorn”.

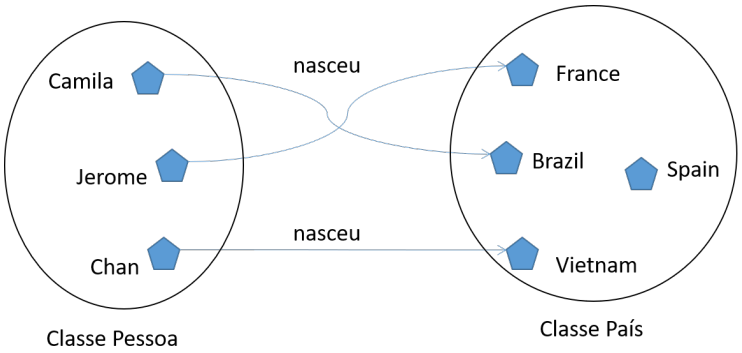


Figura 4 – Exemplo de Propriedade

- **Axioma** - é utilizado para modelar sentenças que são consideradas sempre verdadeiras. Exemplo: Um carnívoro come carne, onde “carnívoro” e “carne” são classes e “come” é uma “relação” entre elas;

- **Hierarquia de classes e propriedades**: Em uma ontologia, pode-se criar uma hierarquia de classes. Por exemplo: veículo é uma superclasse de carro. Também podemos ter hierarquia de propriedades: “temCobertura” é subclasse de “temIngrediente”, de acordo com a ontologia de Pizza.

Na Figura 5 é apresentado um pedaço de uma ontologia de pizza[1], na qual aparecem conceitos e subconceitos, como por exemplo: “SpicyPizza” é um subconceito de “Pizza”. Além disso, são exibidos indivíduos do conceito “Country” que são *France, England, America, Italy e Germany*, destacados pelo símbolo .

Na seção seguinte é apresentada uma descrição sobre a área de engenharia de ontologias que trata do processo do desenvolvimento de ontologias.

[1] *http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library*

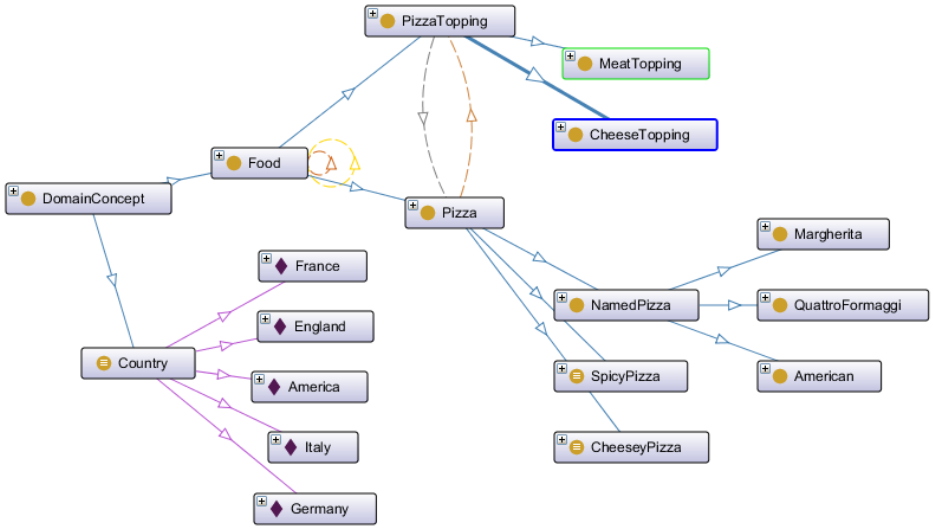


Figura 5 – Pedaço da ontologia de pizza

2.2 ENGENHARIA DE ONTOLOGIAS

Com o avanço da área de ontologias, fez-se necessária a criação de uma área mais ampla denominada EO que, de acordo com (GOMEZ-PEREZ; CORCHO; FERNANDEZ-LOPEZ, 2004), se refere a um conjunto de atividades que considera o processo, o ciclo de vida, os métodos, as metodologias, ferramentas e linguagens que dão suporte ao desenvolvimento de ontologias.

De uma maneira geral, uma metodologia é um conjunto de práticas ou métodos estruturados para atingir um determinado objetivo. Já um método é um processo ou um procedimento ordenado utilizado na engenharia de um produto ou prestação de um serviço ((IEEE, 1995)). Além disso, a principal diferença entre uma técnica e um método é que um método requer uma ordem e o outro não. No entanto, ambos podem fazer parte de uma metodologia.

Com o objetivo de criar ontologias com um certo nível de organização e documentação, existem várias metodologias. A maioria descreve como construir uma ontologia a partir de um modelo e/ou reusar outras ontologias: Methontology ((FERNÁNDEZ-LÓPEZ; GÓMEZ-PÉREZ; JURISTO, 1997)), (LENAT; GUHA, 1989) method, (USCHOLD; KING, 1995) method, On-To-Knowledge ((STAAB et al., 2001)) etc.

Geralmente elas apresentam características comuns, como

- **Ciclo de vida:** identifica o conjunto de etapas pelas quais uma ontologia deve passar durante o seu tempo de vida; descreve as atividades que devem ser executadas em cada etapa e como as etapas são relacionadas; e o
- **Processo de desenvolvimento:** deve ser executado para construir uma ontologia, incluindo: processo de requisitos, com atividades interativas que são direcionadas

para desenvolvimento da especificação dos requisitos da ontologia; processos de concepção, cujo objetivo é desenvolver uma representação coerente e bem organizada da ontologia que atenda à especificação de requisitos; e processo de implementação, que transforma a representação de um modelo em uma linguagem de implementação de uma ontologia.

A seguir, um resumo de algumas metodologias que propõem o uso de questões de competência:

- **Uschold and Gruninger ((USCHOLD; GRUNINGER, 1996)):** Descreve o processo de documentação, de desenvolvimento e validação de uma ontologia a partir de técnicas informais, como o uso de documentos em linguagem natural para técnicas que usam lógica formal. Além disso, considera a representação formal de questões de competência em uma maneira que reconhece axiomas para descrições de uma ontologia;

- **Methontology ((FERNÁNDEZ-LÓPEZ; GÓMEZ-PÉREZ; JURISTO, 1997)):** Visa elucidar as atividades necessárias para criar novas ontologias. Essa metodologia descreve o processo de desenvolvimento de ontologias dividindo em tipos de atividades. Em outras palavras, ela descreve o ciclo de vida do desenvolvimento de uma ontologia a partir dos protótipos e de técnicas específicas para cada atividade realizada. As atividades são orientadas para o desenvolvimento e atividades de apoio. Na fase de especificação, incentiva a criação de um documento de especificação da ontologia, escrito em linguagem natural e usando QCs;

- **NeOn methodology ((SUÁREZ-FIGUEROA; GÓMEZ-PÉREZ; FERNÁNDEZ-LÓPEZ, 2012)):** Surgiu como uma proposta de evolução dos métodos já disponíveis e inclui o desenvolvimento colaborativo e distribuído. Como outras metodologias, esta também introduz a ideia de QC como uma opção viável à identificação dos requisitos de ontologias.

No entanto, a maioria das metodologias descreve QCs superficialmente. QCs são mencionadas na fase de documentação inicial, apoiadas por declarações baseadas em linguagem natural controlada por meio de planilhas. Infelizmente, não é possível avaliar até que ponto a ontologia satisfaz uma QC e como utilizá-las de forma mais automatizada por meio de ferramentas.

Na próxima seção é descrita aplicações de ontologias, como na Web Semântica, uma proposta de uma Web mais inteligente e que utiliza ontologias para alcançar tal objetivo.

2.3 APLICAÇÕES DE ONTOLOGIAS

Ontologias são utilizadas largamente em diferentes areas como Medicina ((ALDOSARI; ALANAZI; HOUSEH, 2017)), Política ((SANTOS; ROVER, 2016)), Geografia[2], Astronomia ((FREY; ACCOMAZZI, 2018)), Direito ((RODRIGUES et al., 2019)), e principalmente dentro da comunidade ciências naturais e biológicas ((BARD; RHEE, 2004)).

2.3.1 Web Semântica

O uso de ontologias aumentou com o surgimento da *Web Semântica* proposta por (BERNERS-LEE; HENDLER; LASSILA, 2001), que tem como objetivo prover uma estrutura semântica às páginas Web, de modo que não apenas os seres humanos, mas também agentes computacionais possam compreender os conteúdos contidos nelas. Um dos objetivos da Web Semântica é criar um ambiente onde agentes de software possam navegar pelos documentos Web realizando tarefas sofisticadas como, por exemplo, fornecer um material de aprendizagem adequado a um usuário que esteja fazendo avaliações em um sistema de aprendizagem de acordo com interações dele com o sistema e necessidades dele. A Figura 6 mostra as camadas da *Web Semântica*, onde uma delas é a camada de ontologia. Uma das tecnologias-chave para tornar possível a *Web Semântica* é a ontologia, que provê um modelo semântico para especificação dos dados, possibilitando um vocabulário comum para troca de informação.

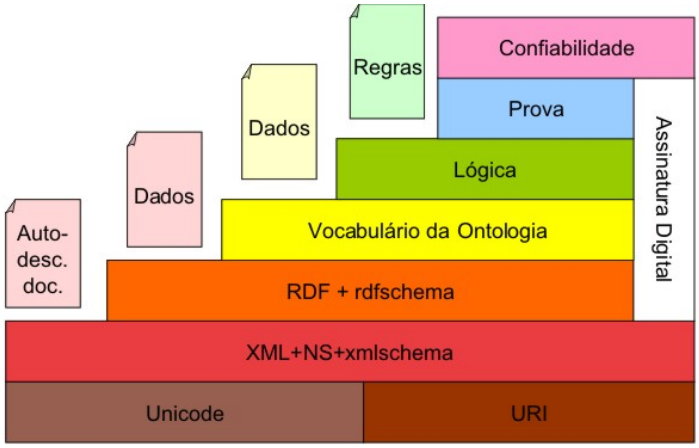


Figura 6 – Camadas da Web Semântica

As ontologias fornecem um vocabulário para a representação do conhecimento, o qual possui uma conceitualização que o sustenta, evitando interpretações ambíguas desse vocabulário ((NEVES, 2006)). Também é possível a reutilização de uma ontologia ou de parte de uma ontologia por outras aplicações que necessitem do mesmo domínio, favorecendo, dessa forma, o compartilhamento de conhecimento.

[2] http://www.obofoundry.org/ontology/geo.html

Para mais detalhes sobre a Web Semântica ver apêndice D.

A linguagem mais expressiva para representar ontologias e recomendada pela w3c é a owl, descrita na próxima seção.

2.3.2 OWL

Como o Resource Description Framework (RDF) é uma linguagem que apresenta uma limitação de expressividade, World Wide Web Consortium (W3C) desenvolveu uma linguagem mais expressiva para Web Semântica, a Ontology Web Language (OWL), que é uma linguagem baseada em lógica de descrição, com o objetivo de descrever conhecimento rico e complexo, geralmente, acerca de um domínio em particular, podendo ser utilizada para criar ontologias.

A atual versão da OWL é a 2.0 desenvolvida a partir de limitações significativas na expressividade da linguagem da versão anterior.

A Figura 7 exibe a estrutura da OWL 2.0. No centro existe uma noção abstrata de ontologia abaixo dois tipos de semânticas disponíveis; e, acima, tipos de sintaxes disponíveis.

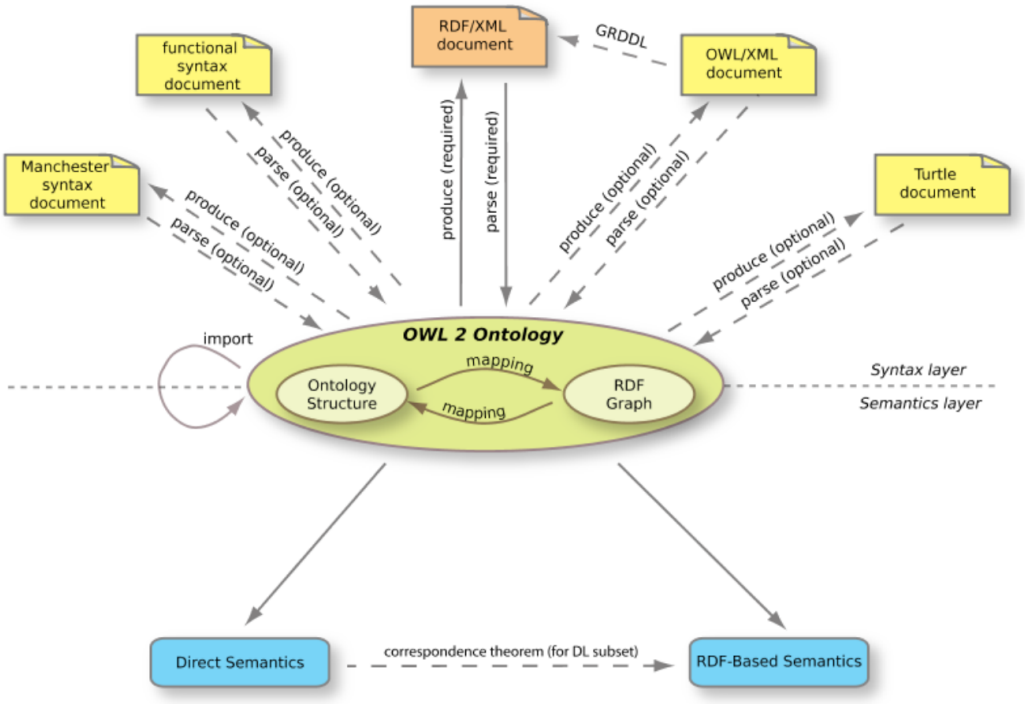


Figura 7 – Estrutura de OWL 2 [Extraído de (W3C, 2012)]

A ontologia pode ser definida utilizando uma estrutura que segue a especificação (W3C, 2012) ou pode ser definida utilizando grafos RDFs. Também é possível fazer um mapeamento entre esses dois tipos de estruturas disponíveis.

Existem várias sintaxes já conhecidas como a RDF/eXtensible Markup Language (XML) e a *Manchester syntax*. Elas têm como propósito armazenar ontologias em OWL

e permutá-las entre ferramentas e aplicações.

Com relação à semântica para RDF Schema, existem a *Direct Semantics* e a *RDF-Based Semantics*. A *Direct Semantics* atribui semântica diretamente às estruturas da ontologia. Assim, temos como resultado uma semântica compatível com o modelo teórico da semântica.

OWL 2.0 oferece 3 tipos de sublinguagens, cada uma com restrições sintáticas, porém cada uma com vantagens a oferecer, dependendo da aplicação a ser utilizada.

OWL 2 EL

É uma linguagem baseada na lógica de descrição $\mathcal{EL}++$ ((BAADER; BRANDT; LUTZ, 2005)), sendo mais interessante para aplicações que utilizam grandes ontologias, ou seja, que contêm um grande número de classes e propriedades, com poder de expressividade garantida ((GRAU et al., 2008)). Por exemplo, ela é suficiente para expressar a ontologia médica SNOMED[3]. As tarefas de raciocínio são efetuadas em tempo polinomial e o seu principal serviço de raciocínio é o de classificação.

Particularmente, os tipos de restrições de classes permitidos são:

- **Quantificador existencial para uma classe (ObjectSomeValuesFrom)**: define uma classe como o conjunto de todos os indivíduos que são conectados por uma propriedade particular para outro indivíduo, o qual é instância de uma certa classe. Por exemplo, o axioma abaixo define uma classe de pais como uma classe de indivíduos que são ligados a uma pessoa pela propriedade "temFilho".

  EquivalentClasses(:Pai ObjectSomeValuesFrom(:temFilho :Pessoa))

- **Quantificador existencial para um indivíduo (ObjectHasValue)**: descreve uma classe de indivíduos que está relacionada a um indivíduo em particular. Por exemplo, o axioma abaixo está relacionando a classe "FilhosFred" com o indivíduo "Breno".

  EquivalentClasses(:FilhosFred ObjectHasValue(:temFilho :Breno))

- **Quantificador existencial para intervalo de dados (DataSomeValuesFrom)**: no exemplo abaixo, é definida uma classe denominada "Adolescente" como todos os indivíduos que têm idade entre 12 e 18.

```
SubClassOf( :Adolescente
  DataSomeValuesFrom( :temIdade
      DatatypeRestriction( xsd:integer
          xsd:minExclusive "12"8sd:integer
          xsd:maxInclusive "18"8sd:integer
      )
```

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

)
)

- **Quantificador existencial para literal (DataHasValue)**: contém todos os indivíduos conectados por uma propriedade a um valor. No exemplo abaixo, define os indivíduos representados por "ind" que tem idade igual a 25.
  DataHasValue(ind:temIdade "25"8sd:integer)

- **Auto restrição (ObjectHasSelf)**: descreve uma classe de indivíduos que estão relacionados a si mesmos. No exemplo abaixo, é definida uma pessoa narcisista que é uma pessoa que ama a si mesmo.
  EquivalentClasses(:Narcisista ObjectHasSelf(:ama))

- **Enumerações envolvendo um individuo único (ObjectOneOf) ou um único literal(DataOneOf)**: ObjectOneOf descreve uma classe para enumerar todos os indivíduos da classe. DataOneOf descreve um *datatype* gerado pelos valores que ele contém.

```
EquivalentClasses(
   :MeusAlunos
         ObjectOneOf( :Maria :Joana :Lucas :Rafael)
)

DatatypeDefinition(
:apartamentos
DataOneOf( "101"8sd:integer "102"8sd:integer "103"8sd:integer "104"8sd:integer)
)
```

- **Interseção de classes(ObjectIntersectionOf)**: descreve a interseção de duas classes.

```
SubClassOf(
   :Mae
         ObjectIntersectionOf( :Mulher :Parente )
)
```

- **Interseção de intervalo de dados(DataIntersectionOf)**: no exemplo abaixo, assumindo a existência de um *datatype* "maiorIdade", definimos um *datatype* chamado "menorIdade" pela exclusão de todos os valores de "maiorIdade" de "idadePessoa".

```
DatatypeDefinition(
   :menorIdade
```

DataIntersectionOf(
:idadePessoa
DataComplementOf(:maiorIdade)
))

Para maiores detalhes visitar o referencial da W3C em <https://www.w3.org/TR/2012/REC-owl2-profiles-20121211/>.

Além da OWL EL utilizada nesse trabalho, existem a OWL QL e OWL RL, para mais detalhes, ver apêndice A.

3 LÓGICA DE DESCRIÇÕES

Lógica de descrições (*Description logics*) é uma família de linguagens de representação de conhecimento que pode ser utilizada para representação de domínios (por exemplo, domínio da medicina, do direito, e de qualquer outro domínio que se faça necessário representá-lo para algum fim). Esta tecnologia teve sua origem no esforço de dar semântica precisa a formalismos anteriores baseados em estrutura de redes, como rede semântica e frames ((FIKES; KEHLER, 1985)). Diferentemente dos seus antecessores, ((BAADER; HORROCKS; SATTLER, 2008)) apresentam uma lógica com semântica formal, portanto não dando margens a ambiguidades, como nos formalismos antecessores (Redes Semânticas e Frames) . Além de consistir num fragmento decidível de First Order Logic (FOL) com predicados com no máximo aridade 2. Como todas as relações são binárias, é possível escrever as sentenças lógicas, sem necessidade de variáveis.

3.1 ALFABETO

O alfabeto é composto por:

- Símbolos de conceitos que são representados por letras ou palavras começando com letra maiúscula, além dos símbolos $\top$ e $\bot$;
- Símbolos de papéis que são representados por letras ou palavras começando com letra minúscula;
- Conectivos lógicos: $\sqcap$, $\sqcup$, $\neg$, $\forall$, $\exists$, $\equiv$, $\sqsubseteq$

Regras de formação:

- Todo símbolo de conceito é um conceito;
- Se A e B são conceitos, as seguintes construções são conceitos: A $\sqcap$B, A $\sqcup$B,$\neg$A
- Se A é um conceito e r um papel, as seguintes construções são conceitos: $\exists r.A$, $\forall r.A$
- Se r é um papel, $\exists r.\top$, $\forall r.\bot$, $\exists r.\bot$, $\forall r.\top$ são conceitos.

3.2 SINTAXE

Uma linguagem de formalismo de representação de conhecimento voltado a domínios, como lógica de descrições tem principalmente, os seguintes elementos:

- Indivíduos: Aurora, banco de dados, etc.

- Conceitos: que podem ser vistos como predicados unários. Por exemplo, Pessoa, disciplina, etc.

- Papéis/ relações binárias: que são vistos como predicados binários. Por exemplo, a relação "estáMatriculado" pode ser utilizada para representar que um aluno Pedro "estáMatriculado" na disciplina Lógica, estaMatriculado(Pedro,Lógica).

Definição 1 *Seja* $\boldsymbol{C}$ *um conjunto de nomes de conceitos e* $\boldsymbol{R}$ *um conjunto de papéis disjuntos de* $\boldsymbol{C}$*. O conjunto de descrições da linguagem* $\mathcal{ALC}$*, sobre* $\boldsymbol{C}$ *e* $\boldsymbol{R}$ *é definido por:*

- *Todo nome de conceito é uma descrição de conceito* $\mathcal{ALC}$*;*

- $\top$ *e* $\bot$ *são descrições de conceitos* $\mathcal{ALC}$ *e* R *é um nome de um papel, então as seguintes descrições também são válidas:*

  - *Conjunção:* $C \sqcap D$
  - *Disjunção:* $C \sqcup D$
  - *Negação:* $\neg C$
  - *Restrição existencial:* $\exists r.C$
  - *Restrição universal:* $\forall r.C$

3.3 SEMÂNTICA

Para fixar o significado de conceitos e papéis, faz-se o uso de uma interpretação que consiste de um conjunto não vazio chamado domínio de interpretação.

Definição 2 *Segundo ((BAADER et al., 2010)) Uma interpretação* $I = (\Delta^I, .^I)$ *consiste de um conjunto não vazio* Δ^I*, chamado de domínio de interpretação, e um mapeamento* $.^I$ *que mapea:*

- *Todo conceito* A *para relação unária* $A^I \subseteq \Delta^I$

- *Todo papel* $r \in R$ *para uma relação binária* $r^I \subseteq \Delta^I x \Delta^I$

Além disso, a semântica de LD assume a suposição de mundo aberto, que consiste que ausência de informação é considerada como algo desconhecido.

Abaixo, alguns construtores da sintaxe LD, com a definição da semântica:

- **Conjunção de conceitos**, representado pelo símbolo $\sqcap$, é interpretado como $(C \sqcap D)^I = C^I \cap D^I$

- **União de conceitos**, representado pelo símbolo $C \sqcup D$, e interpretado como $(C \sqcup D)^I = C^I \cup D^I$

- **Quantificador existencial**, representado pelo símbolo $\mathcal{E}$, e é escrito como $\exists R.C$,e é interpretado como
  $(\exists R.C)^I = \{a \in \Delta^I \mid \exists b.(a,b) \in R^I \wedge b \in C^I\}$

- **Restrição de número**, representado pela letra $\mathcal{N}$, é escrito como $\geq nR$ ou $\leq nR$, e é interpretado como
  $(\geq nR)^I = \{a \in \Delta^I \Big| |\{b|(a,b) \in R^I\}| \geq n\}$

- **Negação de conceito**, representado pelo símbolo $\mathcal{C}$, e é escrito como $\neg C$, e é interpretado como
  $(\neg C)^I = \Delta^I \backslash C^I$

Na próximas seção, será apresentada como representar conhecimento em LD.

3.4 REPRESENTAÇÃO DE CONHECIMENTO EM LD

Em lógica de descrições (LD), a representação de um domínio é constituída de "conceitos", que são termos de um domínio, "propriedades" desses conceitos, e "indivíduos" desse domínio. Além disso, é possível estabelecer relações entre os conceitos, como a relação "É-UM", que permite a criação de uma hierarquia de conceitos, além de axiomas terminológicos.

Um sistema de representação de conhecimento (ver Figura 8) baseado em lógica de descrições contém dois principais componentes:

- **Terminological Axioms (TBox)**: representa a terminologia do conhecimento representado, isto é, um conjunto de axiomas. Além do vocabulário da terminologia que contém conceitos (por exemplo, Aluno) e relações (por exemplo, orienta) entre os conceitos. O conceito de TBox é equivalente a um esquema de um banco de dados.

- **Assertional Axioms (ABox)**: contém o conhecimento extensional, isto é, os indivíduos do domínio, equivalente a noção de "dados" em um banco de dados, por exemplo, o indivíduo Camila que é do tipo Aluno(conceito).

3.4.1 Axiomas terminológicos

Axiomas terminológicos fazem declarações sobre o modo como os conceitos e as propriedades se relacionam entre si. Geralmente terminologias definem hierarquias e equivalência entre conceitos. Os axiomas terminológicos apresentam uma das formas abaixo:

- $C \sqsubseteq D$
- $C \equiv D$

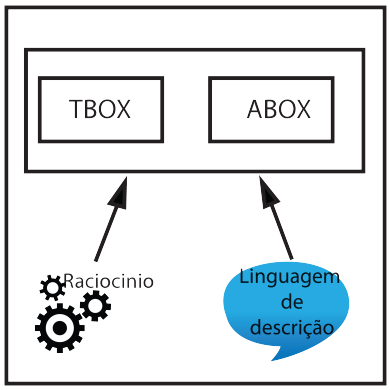


Figura 8 – Framework DL de (BAADER et al., 2010)

Onde C e D são conceitos, e R e S são propriedades. No exemplo abaixo, definimos que uma mulher feliz é uma humana que não é homem, tem menos de dois filhos, que se for casada, será apenas com uma instância de um bom homem, assim como se tiver um trabalho, será apenas com uma instância de um bom trabalho, e por fim, a depender do nível de ensino terá que ser instância de doutorado. Para um melhor entendimento, veja o gráfico 9.

MulherFeliz $\equiv$ Humano $\sqcap$ $\neg$Homem $\sqcap$ $(\forall casada.BomHomem) \sqcap$ $(\forall nivelEstudo.Doutorado) \sqcap \leq 2 temFilho \sqcap \forall$tem.BomTrabalho

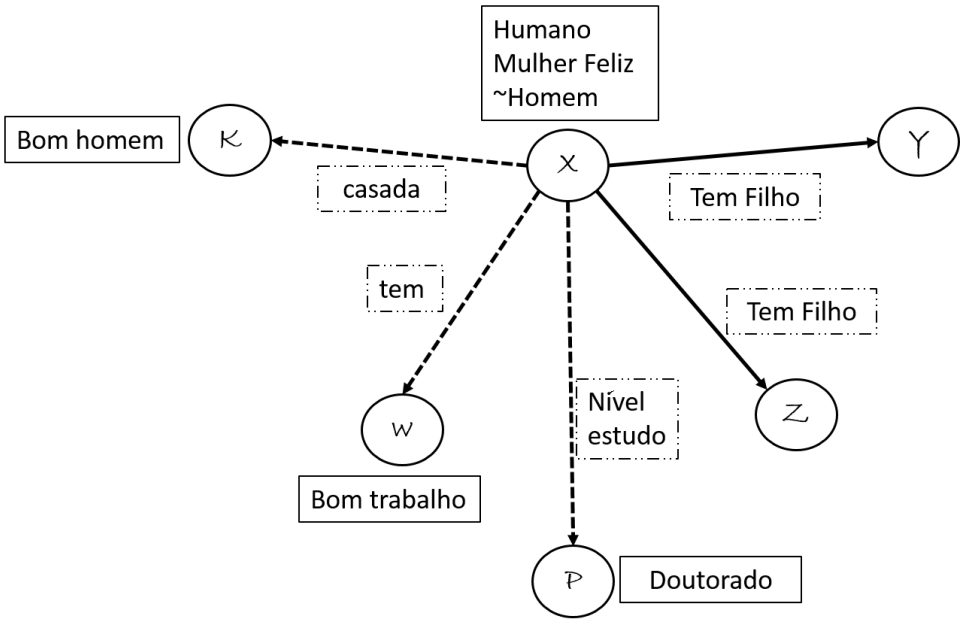


Figura 9 – Detalhamento do exemplo em DL

No gráfico 9 é mostrado o modelo lógico do axioma “MulherFeliz”, onde x é instância de Mulher feliz, y e z são filhos, w é instância de “Bom trabalho”, e k é instância de “Bom

homem".

3.4.2 Tarefas de raciocínio

Um sistema LD também oferece tarefas de raciocínio que consistem em atividades que envolvem manipular conhecimento para gerar novo conhecimento. Geralmente LD é utilizada junto com uma ontologia, a qual pode ser definida como uma representação de um domínio que contém classes (conceitos), propriedades (papéis) e indivíduos. Dentre essas atividades, existe por exemplo, a tarefa de identificar se um conceito é satisfatível, isto é, não contraditório dentro do sistema. Segundo (BAADER et al., 2010) as principais tarefas de raciocínio são:

- **Classificação de conceitos**: determinar relações de hierarquia entre conceitos. Isto é, identificar um conceito que é mais geral que outro. Considere o exemplo abaixo:
  1. Motoristas dirigem veículos
  2. Motoristas de ônibus dirigem ônibus
  3. Um ônibus é um veículo

  Em lógica de descrições:
  $Motoristas \equiv \exists dirigirem.Veiculo$
  $MotoristaOnibus \equiv \exists digirirem.Onibus$
  $Onibus \sqsubseteq Veiculo$

  Uma tarefa de classificação permite inferir a partir do conhecimento acima que $MotoristaOnibus \sqsubseteq Motoristas$

- **Classificação de indivíduos (instâncias)**: determinar se um indivíduo é uma instância de um conceito. No exemplo abaixo, Joana é classificada como indivíduo da classe Mãe e que tem um filho Pedro. Partimos da suposição da existência de uma base de conhecimento que já está descrito o que é pessoa e o que é Mulher. Além disso, toda base de conhecimento em LD parte de nomes de conceitos que não tem uma definição. Assim, nesse caso os conceitos Pessoa e Mulher em algumas terminologias são chamados de conceitos primitivos e em outras de conceitos atômicos.

  $Mãe \sqsubseteq Mulher \sqcap \exists temFilho.Pessoa$
  $Mulher(Joana)$
  $temFilho(Joana, Pedro)$

  $\models$
  $mae(Joana)$

- **Equivalência:** Nós dizemos que dois conceitos são equivalentes, digamos A ≡ B, se $A^I \equiv B^I$ para todas as interpretações de $\mathcal{I}$. Por exemplo,
  ∀ temFilho.Mulher ⊓ ∀ temFilho.Estudante ≡ ∀ temFilho.(Mulher ⊓ Estudante)

- **Verificação de satisfatibilidade**: identificar conceitos insatisfatíveis, que são aqueles os quais não podem ter nenhum indivíduo. São conceitos equivalentes a um conjunto vazio ((KALYANPUR et al., 2005)).

  $PizzaVegana \sqsubseteq Pizza \sqcap \exists temCobertura.Tomate \sqcap \exists temCobertura.Cebola$
  $\exists temCorbetura.CarneDeJaca \sqcap \exists temCobertura.Queijo \sqcap$
  $\neg(\exists temCobertura.AlimentosDeOrigemAnimal)$
  $Queijo \sqsubseteq AlimentosDeOrigemAnimal$

  $\models$
  $PizzaVegana \models \bot$

  No exemplo acima, PizzaVegana é um conceito insatisfatível, pois em sua definição consta que ela não apresenta alimentos de origem animal, porém ela contém uma alimento de origem animal que é queijo.

- **Verificação de consistência**: uma ontologia é inconsistente quando apresenta indivíduos de classes insatisfatíveis.

  Por exemplo, uma ontologia pode se tornar inconsistente, por meio da instanciação de duas classes as quais são disjuntas. Digamos que uma ontologia contém um indivíduo chamado mimosa do tipo vaca, e vaca é do tipo herbívoro(conceito) e carnívora(conceito), porém, carnívora e herbívoro são classes disjuntas, fazendo com que nenhum indivíduo possa ser carnívoro e herbívoro ao mesmo tempo, como mostra o exemplo em LD abaixo.
  Vaca ⊑ Animal
  Carnivora ⊑ ¬ Herbivoro
  Vaca ⊑ Carnivora ⊓ Herbivoro
  $\models$
  $\bot$

Além disso, um sistema de representação de conhecimento geralmente faz parte de um sistema maior, interagindo com outros componentes, como um sistema de consultas que pode utilizar regras para recuperar ou modificar a base de conhecimento.

Na próxima seção serão definidos formalmente conceitos apresentados nesse capítulo.

3.4.3 Deinições Formais

Segundo (BAADER et al., 2010):

Definição 3 *(**TBox**). Sejam C,D dois conceitos, uma TBox 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 T são denominadas axiomas terminológicos. T é consistente se existe uma interpretação I que satisfaz $C^I \subseteq D^I$ para cada $C \sqsubseteq D \in T$ e que satisfaz $C^I = D^I$ para cada $C \equiv D \in T$. Nesse caso, I é denominado modelo de T.*

Definição 4 *(**ABox**). Sejam a,b dois indivíduos, C um conceito e R um papel, uma ABox 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 A são denominadas asserções. A é consistente em relação a TBox T se existe um modelo I de T no qual $a^I \in C^I$ é satisfeito para cada $a : C$ em A e (a^I,b^I) $\in R^I$ é satisfeito para cada $\langle a, b\rangle : R \in A$. Nesse caso, I é denominado modelo de A.*

Definição 5 *(**Base de conhecimento**). Uma base de conhecimento S é definida como uma tupla $\sum$ = (T,A), onde T é uma TBox e A é uma ABox. Uma interpretação I satisfaz S se I for um modelo de T e se I for um modelo de A.*

Definição 6 ***(Satisfatibilidade de conceitos)**. checa se $S \not\models C \equiv \bot$, ou seja, um conceito C é dito como satisfatível em relação a uma base de conhecimento $\sum$, se e somente se existe um modelo I de $\sum$ no qual $C^I \neq \emptyset$.*

Definição 7 ***Subsunção(Classificação de conceitos)**. checa se $\sum \models C \sqsubseteq D$, ou seja, D subsume C, se e somente se todos os modelos I de $\sum$ satisfizerem $C^I \subseteq D^I$.*

Definição 8 ***(Verificação de instância).** seja a um indivíduo, checa se $\sum \models a : C$, ou seja, se a é uma instância do conceito C. A checagem é positiva se $a^I \in C^I$ para todos os modelos I de $\sum$.*

Na próxima seção, discorreremos das principais linguagens LD, sua sintaxe e semântica.

3.5 LINGUAGENS

Na literatura existem várias linguagens de descrições, cada uma com um nível de expressividade diferente. Elas são diferenciadas pelos construtores que cada uma contém. As linguagens de descrições apresentam conceitos e papeis atômicos que são descrições elementares, e a partir dessas descrições elementares somos capazes de construir descrições complexas que equivalem a construtores de conceitos e construtores de papéis.

A seguir, descreveremos as linguagens utilizadas neste trabalho.

3.5.1 Linguagem $\mathcal{ALCN}$

Um outro tipo de linguagem LD é alcn que uma extensão de $\mathcal{ALC}$ acrescido de restrição de número. $\mathcal{ALCN}$ é composta pelos construtores: negação ($\neg$), conjunção($\sqcap$), disjunção ($\sqcup$), quantificador existencial ($\mathcal{E}$), quantificador universal ($\forall$), e restrições de número como mostra a Tabela 1.

Tabela 1 – Sintaxe x Semântica de $\mathcal{ALCN}$.

Construtor	**Sintaxe**	**Semântica**
Indivíduos		
Nome do Indivíduo	a	a^I
Papéis		
Relação	R	R^I
Conceitos		
Conceito	A	A^I
Intersecção	$\mathsf{C} \sqcap \mathsf{D}$	$\mathsf{C}^I \cap \mathsf{D}^I$
União	$\mathsf{C} \sqcup \mathsf{D}$	$\mathsf{C}^I \cup \mathsf{D}^I$
Complemento	$\neg\mathsf{C}$	$\Delta^I \setminus \mathsf{C}^I$
Top Concept	$\top$	Δ^I
Bottom Concept	$\bot$	$\{\ \}$
Restrição Existencial	$\exists\mathsf{R}.\mathsf{C}$	$\{\mathsf{x} \in \Delta^I \mid \exists\ \mathsf{y}, (\mathsf{x},\mathsf{y}) \in \mathsf{R}^I \text{ and } \mathsf{y} \in \mathsf{C}^I\}$
Restrição Universal	$\forall\mathsf{R}.\mathsf{C}$	$\{\mathsf{x} \in \Delta^I \mid \forall\ \mathsf{y}, (\mathsf{x},\mathsf{y}) \in \mathsf{R}^I \Rightarrow \mathsf{y} \in \mathsf{C}^I\}$
Restrição Numérica	$\geq n\mathsf{R}.\mathsf{C}$	$\{\mathsf{x} \in \Delta^I \mid \exists$ ao menos $\mathsf{n}\ \mathsf{y}, (\mathsf{x},\mathsf{y}) \in \mathsf{R}^I \text{ and } \mathsf{y} \in \mathsf{C}^I\}$
	$\leq n\mathsf{R}.\mathsf{C}$	$\{\mathsf{x} \in \Delta^I \mid \exists$ no máximo $\mathsf{n}\ \mathsf{y}, (\mathsf{x},\mathsf{y}) \in \mathsf{R}^I \text{ and } \mathsf{y} \in \mathsf{C}^I\}$

3.5.2 Linguagem $\mathcal{SROIQ}$

A SROIQ ((KROTZSCH; SIMANCIK; HORROCKS, 2012)) apresenta a maior expressividade com relação as outras linguagens decidíveis e possui as seguintes novas funcionalidades:

- papéis disjuntos;
- papéis reflexivos e irreflexivos;
- asserção de papéis negados;
- hierarquia complexa de papéis;
- papel universal;
- conceitos self.

Uma ontologia em SROIQ é um conjunto de axiomas sobre a assinatura $\sum = (N_C, N_R, N_S, N_O)$, onde N_C é um conjunto de nomes de conceitos (símbolos de predicado unários), N_R é conjunto de papéis ou nomes de propriedades (símbolos de predicados binários); N_S é o conjunto de papéis simples ou nomes de propriedades; N_O é o conjunto de nome de indivíduos. Os conjuntos são mutuamente disjuntos, exceto por N_S, uma vez que $N_R \supseteq N_S$. O conjunto de papéis (sobre $\sum$) é R = $N_R \cup \{r^- \mid r \in N_R\}$; e o conjunto de papéis simples é equivalente a S = $N_S \cup \{r^- \mid r \in N_S\}$. A cadeia de papéis é uma expressão $r_1 \circ ... \circ r_n$, com n ≥ 1 e $r_i \in \mathbf{R}$. A função inv($\circ$) é definido sobre os papéis por inv(r) = r^- e $inv(r^-)$ = r, r $\in \mathbf{R}$, e uma extensão das cadeias de papéis por $inv(r_1 \circ ... \circ r_n) = inv(r_n,) \circ ... \circ inv(r_1)$.

Na Tabela 2 é apresentado a sintaxe e semântica de SROIQ.

Tabela 2 – Sintaxe x Semântica de $\mathcal{SROIQ}$.

Construtor	**Sintaxe**	**Semântica**
Indivíduos		
Nome do Indivíduo	a	a^I
Papéis		
Relação	R	R^I
Relação Inversa	R^-	{<x,y>|<y,x> $\in R^I$}
Relação Universal	U	Δ^I x Δ^I
Conceitos		
Conceito	A	A^I
Intersecção	C $\sqcap$ D	$C^I \cap D^I$
União	C $\sqcup$ D	$C^I \cup D^I$
Complemento	$\neg$C	$\Delta^I \setminus C^I$
Top Concept	$\top$	Δ^I
Bottom Concept	$\bot$	{ }
Restrição Existencial	$\exists$R.C	{x $\in \Delta^I$ | $\exists$ y, (x,y) $\in R^I$ and y $\in C^I$}
Restrição Universal	$\forall$R.C	{x $\in \Delta^I$ | $\forall$ y, (x,y) $\in R^I \Rightarrow$ y $\in C^I$}
Restrição Numérica	$\geq n$R.C	{x $\in \Delta^I$ | $\exists$ ao menos n y, (x,y) $\in R^I$ and y $\in C^I$}
	$\leq n$R.C	{x $\in \Delta^I$ | $\exists$ no máximo n y, (x,y) $\in R^I$ and y $\in C^I$}
Reflexividade Local	$\exists$R.*Self*	{x | <x,x> $\in R^I$}
Nominais	{a}	$\{a\}^I$

3.5.3 Outras LDs

Existem outras linguagens expressivas em que é possível ter papéis transitivos, composição e disjunção de papéis, restrições de cardinalidades qualificadas, dentre outras. Como nesta tese ficaremos restrito a $\mathcal{ALCN}$, não iremos descrever essas linguagens que culminam com a mais expressiva da web semântica, que é a $\mathcal{SROIQ}$ ((RUDOLPH, 2011)). Para mais detalhes das outras linguagens consultar o apêndice E.

4 MODULARIZAÇÃO DE ONTOLOGIAS: ESTADO DA ARTE

Este capítulo apresenta uma introdução a área de modularização de ontologias, e trabalhos encontrados na literatura até o final da escrita dessa tese. Além disso, no final é apresentado uma discussão sobre os trabalhos apresentados.

4.1 INTRODUÇÃO

Na maioria das metodologias, o desenvolvimento de ontologias apresenta etapas complexas como manutenção, avaliação e integração. Essas etapas apresentam uma certa complexidade devido a problemas como escalabilidade e interoperabilidade de ontologias. Com o avanço da *Web Semântica*, tem surgido a necessidade de ontologias grandes o suficiente para conter todo o conhecimento necessário para determinada aplicação. Porém, isso gera desafios para, principalmente, ferramentas de raciocínio.

OWL contém um construtor *owl:imports* que possibilita importar uma ontologia dentro de outra, porém é um mecanismo ineficaz, pois provê alguma modularidade sintática, mas não lógica.

Noções importantes da Engenharia de Software foram agregadas à Engenharia de ontologias com o propósito de auxiliar os profissionais no desenvolvimento de ontologias, como o conceito de modularização.

Em engenharia de software, segundo (RIBEIRO; JUNIOR; BORBA, 2010)

> "Modularização é uma técnica de software usada para decompor o software em módulos. Cada módulo do software é responsável por implementar uma funcionalidade relevante, separando esse módulo das outras partes do software. Dessa forma, melhorando a capacidade de compreensão e manutenção do módulo".

Na engenharia de ontologias, encontramos as definições abaixo.
Segundo (D'AQUIN et al., 2007)

> "A partir da perspectiva da engenharia de ontologias, modularização deveria ser considerada como um meio de estruturar ontologias, significando que a construção de uma ontologia grande deveria ser baseada na combinação de componentes de conhecimento independentes, autocontidos e reusáveis. "

E (GRAU et al., 2005) definem módulo como

> "Uma entidade que é o subconjunto mínimo de axiomas da ontologia que capturam seu significado necessário e suficiente, portanto, o mínimo conjunto de

axiomas que são necessários para compreender, processar, evoluir e reutilizar a entidade."

Segundo (STUCKENSCHMIDT; PARENT; SPACCAPIETRA, 2009) a modularização de ontologias se propõe a resolver problemas, como:

- **Escalabilidade**
  Quanto maior o tamanho de uma ontologia (número de instâncias e axiomas), o grau de eficiência em algumas tarefas, como de raciocínio e consulta, decresce. A complexidade dessas tarefas é devido ao número de axiomas e instâncias, presentes na ontologia, que precisam ser analisados para chegar a um resultado.

  O principal objetivo do uso de modularização de ontologias em relação à escalabilidade é manter um nível de performance aceitável no oferecimento de serviços de raciocínio, processamento de consultas, ferramentas de visualização entre outros serviços ((STUCKENSCHMIDT; PARENT; SPACCAPIETRA, 2009)).

- **Gerenciamento da complexidade**
  Quanto maior é a ontologia, mais difícil é assegurar a qualidade e acurácia do projeto da ontologia. Além disso, um diferente problema, que é garantir a corretude, surge com relação a raciocinadores $\mathcal{LD}$ para checar a consistência de ontologias. Isso se deve a quantidade de tempo e espaço necessário para um raciocinador conseguir realizar uma atividade em tempo hábil, ou seja, em tempo polinomial determinístico;

- **Reusabilidade**
  É um dos aspectos mais significativos em Engenharia de Software. Ela é utilizada, por exemplo, no desenvolvimento de bibliotecas de software que são reusadas em várias aplicações. Reusabilidade na modularização de ontologias pode ser aplicada em duas abordagens:

  1. Criar módulos a partir da decomposição de uma grande ontologia;
  2. Criar uma ontologia a partir da composição/integração de módulos existentes e mais especializados.

4.1.1 Propriedades Formais da Modularização de Ontologias

Existem muitas definições de modularização e de módulo de uma ontologia. De fato, é difícil haver apenas uma definição genérica, já que as propriedades desejadas de um módulo dependem da aplicação que se pretende desenvolver ((KONEV et al., 2009)).

(KONEV et al., 2009) propuseram propriedades formais da modularização para auxiliar os leitores em escolher a definição correta de modularização. Para isso, inicialmente, estabeleceram a definição de módulo, apoiada em noções, como inseparabilidade.

(KONEV et al., 2009) tomaram como partida que um módulo "é parte de um sistema complexo que funciona independentemente do sistema" e tentaram definir o que é um módulo com relação a ontologias, ou seja, "especificar o que significa um módulo funcionar independentemente da ontologia".

Eles consideraram que uma ontologia O pode ser considerada como "uma caixa preta que provê respostas para perguntas sobre um vocabulário de interesse S". Sendo assim, para definir o que é um módulo, primeiro é necessário estabelecer uma linguagem de consulta denominada **Q** e um vocabulário de interesse denominado **S**.

Os autores chegaram à conclusão que "dados uma ontologia e seus módulos, nós podemos dizer que um módulo funciona independentemente da ontologia se, para qualquer consulta construída, utilizando uma linguagem específica e um vocabulário associado com um módulo, obtemos a mesma resposta quando consultamos o módulo (isoladamente) e a ontologia inteira".

Essas definições estão intimamente ligadas à noção de inseparabilidade que é largamente utilizada em outras definições de modularização.

Definição 9 ***(Inseparabilidade e estensão conservativa)((KONEV et al., 2009)).*** *Seja* QL *uma linguagem de consulta,* O_1*,* O_2 *ontologias,* S *uma assinatura, que consiste no vocabulário de nomes das entidades da ontologia e* sig *a assinatura da consulta que possui o vocabulário S. Nós dizemos que*

- O_1 *e* O_2 *são* ***S-inseparáveis*** *com relação a* QL *e definido como* $O_1 \approx_S^{QL} O_2$*, se e somente se para todo* $\varphi \in QL$ *com* $sig(\varphi) \subseteq S$*, nós temos* $O_1 \models \varphi$ *se e somente se* $O_2 \models \varphi$*.*

  O conceito de inseparabilidade consiste em dada uma linguagem de consulta QL e um vocabulário S, duas ontologias O1 e O2 são S-inseparáveis com relação a QL, se elas dão as mesmas respostas as consultas em QL sob S.

- O_2 *é uma* ***Extensão conservativa (S-conservative extension)*** *de* O_1 *com relação a* QL*, se e somente se* $O_2 \subseteq O_1$*, e* O_1 *e* O_2 *são S-inseparáveis com relação a* QL*. Além disso, se* $S = sig(O_1)$*, então nós dizemos que* O_2 *é uma extensão conservativa de* O_1 *com relação a* QL*.*

  Uma ontologia O2 é uma estensão conservativa de O1, se e somente se, O2 está dentro de O1, além disso O1 e O2 são S-inseparáveis com relação a uma mesma linguagem de consulta denominada QL.

Assim sendo, módulo é um subconjunto de uma ontologia, que é S-inseparável da ontologia, com relação a uma linguagem de consulta *QL* e a uma assinatura S associada ao módulo.

Com relação à complexidade do problema de decisão acerca de inseparabilidade e estensão conservativa, os autores chegaram ao seguinte teorema:

Definição 1 0 *Seja L uma das linguagens de LDs:* $\mathcal{ALC}$, $\mathcal{ALCI}$, $\mathcal{ALCQ}$ *e* $\mathcal{ALCQI}$. *Então o problema* S-inseparabilidade *para (*L,QL*) é* 2-ExpTime-complete. *Além disso, o problema de* S-conservatividade *também é* 2-ExpTime-complete.

Para ($\mathcal{ALCQIO}$, $\mathcal{QL}_{\mathcal{ALCQIO}}$) o problema *S-inseparabilidade* e *S-conservatividade* são indecidíveis.

Na seção a seguir serão descritas algumas das métricas encontradas na literatura. As métricas visam identificar e caracterizar as estruturas recorrentes (padrões) em ontologias organizadas de forma modular, tal como distribuição de classes e propriedades. Com isso, auxilia a determinar se a ontologia está bem modularizada de acordo com as métricas.

4.1.2 Métricas

No artigo (ABB'ES et al., 2012) que tem como contribuição prover padrões para caracterizar ontologias modulares, métricas similiares às encontradas na engenharia de software ((Allen; Khoshgoftaar; Chen, 2001)), os autores adotam um conjunto de métricas:

- **Tamanho do módulo:** número de classes e propriedades;
- **Coesão do módulo:** enfatiza a consistência interna de um módulo. As seguintes métricas são definidas abaixo:
  - **Coesão da hierarquia de classes (*Hierarchical Class Chesion(HCC)):*** O número de ligações hierárquicas entre classes, direta ou indiretamente.

    $$\text{HCC} = \frac{2*(NdHC+NidHC)}{NC^2-NC}$$

    onde NdHC: número de relações diretas de hierarquia entre classes, NidHC: número de relações indiretas de hierarquia entre classes, e NC: número de classes.
  - **Coesão das relações (Role Cohesion(RC))**: número de ligações hierárquicas direta ou indiretamente entre papéis.

    $$\text{RC} = \frac{2*(NdR+NidR)}{NRoles^2-NRoles}$$

    onde, NdR: número de relacionamentos diretos entre classes, NidR= número de relacionamentos indiretos entre classes, e NRoles: número de papéis
  - **Coesão de propriedade de objeto (*Object Property Cohesion(OPC):)*** o número de classes as quais têm associação através de um *object property* em particular(domínio e imagem).

    $$\text{OPC} = \frac{2*\sum_{i=1}^{NRoles} NdC(r_i)*NrC(r_i)}{NRoles*(Nc^2-NC)}$$

onde, $NdC(r_i)$: número de classes da ontologia do domínio da relação r_i, $NrC(r_i)$: número de classes da ontologia na imagem da relação r_i, NC: número de classes, e NRoles: número de relações.

– **Coesão (*Cohesion*)** é computada como:

$$\textbf{Coesão} = \frac{\alpha * HCC + \beta * RC + \sigma * OPC}{\alpha + \beta + \sigma}$$

onde, segundo (ABB'ES et al., 2012) α,β e σ significam o nível de impacto da coesão de cada tipo de hierarquia de classe, relacionamentos ou *object property.*

- **Acoplamento**: envolve aspectos das relações entre os módulos e mede a interdependência entre eles. Estima a interdependência de diferentes módulos. As seguintes métricas são definidas abaixo:

  – **Dependência na hierarquia de classes (*Hierarchical class dependency***
  (HCD)): o número de todas as relações de hierarquia de classes direta ou indiretamente com ontologias ou módulos externos.

$$\textbf{HCD} = \frac{1}{2} * (\frac{NedHC}{NdHC} + \frac{NeidHC}{NidHC})$$

onde, NedHC: número de dependência de hierarquia de classes diretas entre classes locais e classes externas, e NeidHC = número de dependência de hierarquia de classes indiretas entre classes locais e classes externas.

– **Dependência na hierarquia de papéis (*Hierarchical role dependency*** (HRD)): número de todas as relações de hierarquias de relacionamentos diretos ou indiretos com ontologias ou módulos externos.

$$\textbf{HRD} = \frac{1}{2} * (\frac{NdHR}{NdHR} + \frac{NeidHR}{NidHR})$$

onde, NdHR: número de relacionamentos diretos dependentes entre classes locais e classes externas, e NeidHR: número de relacionamentos indiretos dependentes entre classes locais e classes externas.

– **Dependência nos relacionamentos do tipo Object Property (*Object property dependency(OPD)*):** número de relacionamentos que associa classes externas com classes locais.

$$\textbf{OPD} = \frac{NeRoles}{NRoles}$$

onde, NeRoles: número de todos os relacionamentos que têm uma classe externa no domínio ou imagem deles, NRoles: número de todos os papéis existentes na ontologia.

- **Dependência entre axiomas (*Axiom Dependency(AD)*)**: uma classe ou um relacionamento é associada com um elemento ontológico externo através da inclusão de um axioma.

$$\mathbf{AD} = \frac{\sum_{i=1}^{NAxioms} externalAssociationNumber(axm_i)}{\sum_{i=1}^{NAxioms} LS(axm_i * RS(axm_i))}$$

LS(axm):): o tamanho dos lados esquerdos do axioma axm, RS(axm): o tamanho dos lados direitos do axioma axm, LSE(axm): o número de elementos externos nos lados direitos do axioma axm, RSE(axm): número de elementos externos nos lados esquerdos do axioma axm, e *externalAssociationNumber(axm)*: o número de todos os elementos de ontologias ou módulos externos que são associados através do axioma axm com os elementos internos. **externalAssociationNumber(axm)**: $LSE(axm_i) * RS(axm_i) + LS(axm_i) * RSE(axm_i) - LSE(axm_i) * RSE(axm_i)$.

4.2 ABORDAGENS DE MODULARIZAÇÃO DE ONTOLOGIAS

Como na engenharia de software ((PARNAS, 1972)), basicamente, existem três tipos de abordagens:

- **Composição**: criação de uma ontologia pela junção e conexão de módulos externos. A Figura 10 exibe a formação de uma ontologia de câncer de pele utilizando composição de módulos oriundos de duas ontologias diferentes. Formalmente, podemos definir considerando um conjunto de módulos $M_1, ..., M_n$, todos disjuntos, e a junção desses módulos dá origem a uma nova ontologia O;

- **Decomposição(Particionamento)**: quando se decompõe uma ontologia em módulos. O processo de particionamento consiste em dividir o conjunto de axiomas de uma ontologia O em um conjunto de módulos $M_1, ..., M_n$, todos disjuntos, tal que cada M_i é uma ontologia e a união de todos os módulos é semanticamente equivalente à ontologia O original;

- **Extração**: quando se extrai um ou mais módulos de uma ontologia. Na figura 12 é mostrada a extração de um fragmento de uma ontologia sobre células do corpo humano. Formalmente, dada uma ontologia O, podemos extrair um conjunto de axiomas de O, e chamar de módulo M, considerando as propriedades formais de modularização (ver seção 4.1.1).

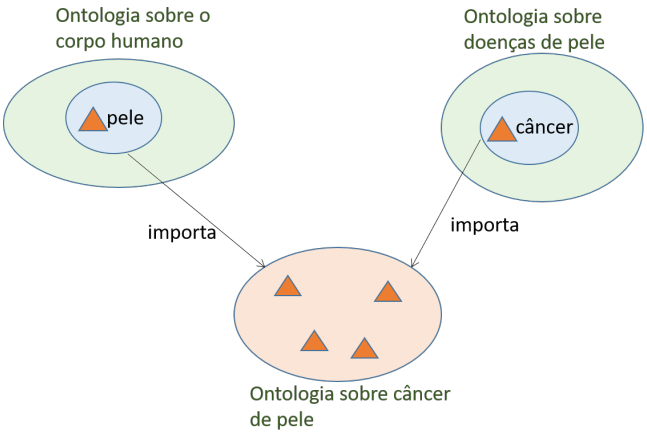


Figura 10 – Exemplo de composição de módulos

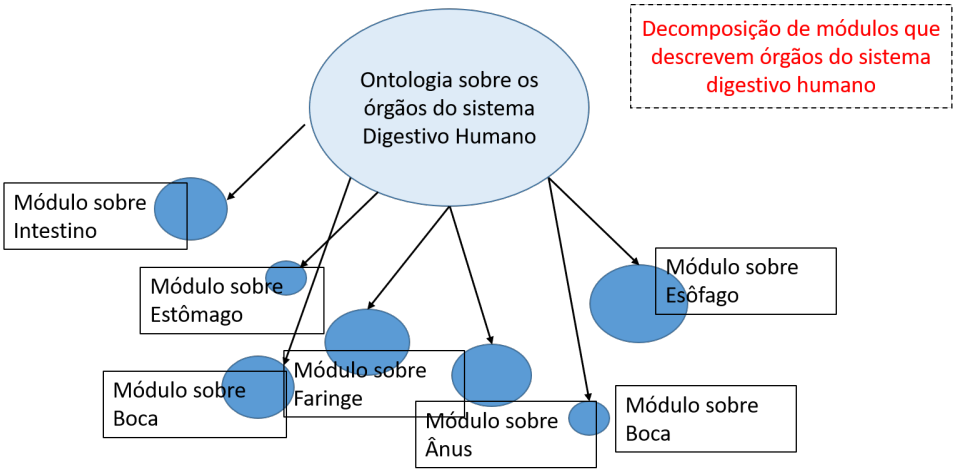


Figura 11 – Exemplo de decomposição de módulos

Na Figura 11 é mostrada uma decomposição de uma ontologia sobre o corpo humano. Nesse exemplo, cada círculo na imagem pode ser considerado um módulo sobre sistema respiratório, ou sobre células, ou sobre sistema digestivo etc.

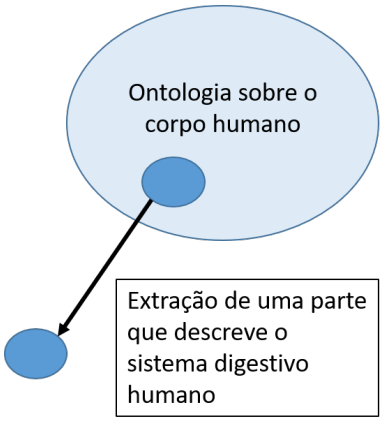


Figura 12 – Exemplo de extração de módulos

4.3 TRABALHOS DE MODULARIZAÇÃO DE ONTOLOGIAS

Nas próxima seções serão apresentados alguns trabalhos de modularização de ontologias.

4.3.1 E-connections

É um trabalho pautado na composição por meio de ligações denominadas $\mathcal{E}$-*connections*. $\mathcal{E}$-*connections* ((GRAU; PARSIA; SIRIN, 2009))) é uma linguagem proveniente da combinação de lógicas decidíveis, de modo que essa combinação continue decidível, apesar do aumento da expressividade.

$\mathcal{E}$-*connections* surgiu a partir de duas características necessárias e que não estavam presentes na OWL (*Ontology Web Language*):

1. Como representar e raciocinar com ontologias distintas, porém conectadas?
2. E como reusar e compartilhar ontologias grandes de maneira eficaz? Além disso, quanto maior a ontologia, mais difícil é utilizar raciocinadores nela.

Uma modularização com $\mathcal{E}$-*connections* é um conjunto de ontologias conectadas que contém informações sobre classes, instâncias e axiomas como uma ontologia OWL convencional, porém apresenta também um novo construtor denominado *link property* que é responsável pela conexão entre as diferentes ontologias.

Por exemplo, considere uma ontologia de doenças infecciosas e uma ontologia de insetos. Digamos que queremos uma ontologia que contenha as doenças às quais cada inseto está relacionado. Por exemplo, a doença Zika está relacionada ao mosquito Aedes aegypti. Para isso, poderemos ter uma link property chamada "causaDoença" para conectar as duas ontologias e, assim, dizer que o mosquito Aedes aegypti causaDoença Zika, como mostrado na Figura 13.

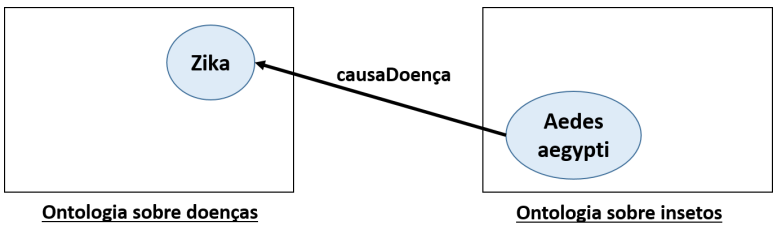


Figura 13 – LinkProperty CausaDoença

4.3.2 Package-based Description Logic(P-DL)

Trabalho que se baseia no reuso através da composição de módulos. P-DL ((BAO et al., 2009)) apresenta um formalismo que suporta reuso contextual de conhecimento a partir de módulos, que podem ser advindos de diversas ontologias. A linguagem resultante desse

trabalho, criada pelo autor, consiste na lógica de descrição com mais expressividade que $\mathcal{ALCN}$ e apresenta as características abaixo ((BAO et al., 2009)):

- Permite que um módulo use um subconjunto da linguagem proposta no trabalho;
- Provê modelagem de domínios mais flexíveis;
- Permite a contextualização do conhecimento reusado;
- Garante a monotocidade de mecanismos de raciocínio;
- Evita dificuldades na tarefa de raciocínio como falta de suporte para reusabilidade transitiva.

Em P-DL, uma ontologia é composta por um conjunto de módulos, denominados pacotes. A sintaxe de P-DL apresenta que um conjunto de conceitos definidos em um pacote seja utilizado em outros pacotes, e os conceitos importados possam ser utilizados na construção de novos pacotes. A diferenciação de um conceito de um pacote em relação aos outros acontece quando utiliza um identificador antes de cada conceito usando um ":". Considere o exemplo de (BAO et al., 2009):

Ontologia sobre pessoas P1:
$\neg 1 : Homem \sqsubseteq 1 : Mulher$
$1 : Homem \sqsubseteq 1 : Pessoa$
$1 : Mulher \sqsubseteq 1 : Pessoa$
$1 : Garoto \sqcup 1 : Garota \sqsubseteq$ 1:Criança
$1 : Esposo \sqsubseteq 1 : Homem \sqcup \exists 1 : casadoCom.1 : Mulher$
$1 : Esposa \sqsubseteq 1 : Mulher \sqcup \exists 1 : casadoCom.1 : Homem$

Considere que a ontologia Empresa P2 importe algumas partes de conhecimento da ontologia Pessoa P1:
$2 : Empregado \sqsubseteq 1 : Pessoa$
$2 : Empregado \equiv \exists 2 : contrata.1 : Pessoa$
1:Criança $\sqsubseteq \neg_2 2 : Empregado$
$2 : OportunidadeEmpregoIgual \sqsubseteq \exists 2 : contrata.1 : Homem \sqcap \exists 2 : emprega.1 : Mulher$

Perceba que, nesse exemplo, foi possível utilizar vários construtores, inclusive especialização e generalização de conceitos.

4.3.3 ModOnto

Este trabalho ((BEZERRA et al., 2008)) apresenta uma ferramenta que incorpora uma abordagem para a modularização de ontologias, herdando, basicamente, dois dos principais princípios da orientação a objetos da engenharia de software que são encapsulamento e

ocultação de informações. O que motivou os autores para seguir nessa direção é o fato de que a maioria das abordagens para modularização tenta resolver o problema através de ligações entre ontologias (ou módulos), ao invés de construir módulos que possam encapsular partes externas de ontologias (ou outros módulos) e ser gerenciados mais facilmente.

O objetivo é fornecer uma definição e composição dos módulos (ou ontologias). Seguindo essa perspectiva, ontologias ou módulos devem ser vistos como blocos de construção, pequenas unidades de conhecimento (ou módulos), projetados para a composição com outros módulos.

A abordagem oferece facilidades que dão suporte para que módulos possam ser especificados, de tal forma que evite a dependência a partir de uma ontologia ou módulo particular. Isso permite que módulos exportados possam ser substituídos por outros com conteúdo semelhante. Nessa abordagem, foi desenvolvida uma linguagem formal com sintaxe e semântica para a implementação de composição de módulos.

Foi desenvolvida uma ferramenta em Java que implementa essa abordagem e que possui os seguintes componentes:

- **Module API:** responsável por lidar com o gerenciamento da linguagem, como criação de módulos;
- **Module checker:** como a ontologia pode evoluir e podem surgir erros na definição dos módulos, o checker verifica se todas as definições necessárias para um módulo estão presentes em seus módulos importados ou nos seus próprios módulos;
- **Module Extractor:** responsável por extrair entidades de uma ontologia;
- **Module Library:** a biblioteca é um repositório de módulos "*off-the-shelf*", que irá ajudar os desenvolvedores a escolher os seus módulos de maneira apropriada;
- **Module Linker:** é uma ferramenta gráfica para construir módulos usando outros módulos. Isso significa interagir com a biblioteca de módulos para obter módulos importados e suportar a especificação de alinhamento necessário entre eles;
- **Module Reasoner:** possibilita fazer consultas em módulos.

Na Figura 14 é ilustrada a construção de um módulo da ferramenta por meio da junção de entidades (classes, propriedades e indivíduo) importadas e criadas.

As principais contribuições desse trabalho são:

1. Verificação semântica do módulo que está sendo criado. Essa verificação é realizada de uma forma muito flexível; os usuários podem escolher entre duas semânticas, a semântica em Lógica Descritiva (LD) que é própria de ontologias habituais, ou a semântica de LD baseada em IDDL ((ZIMMERMANN, 2007)), que é adequada para ontologias em LD distribuídas;

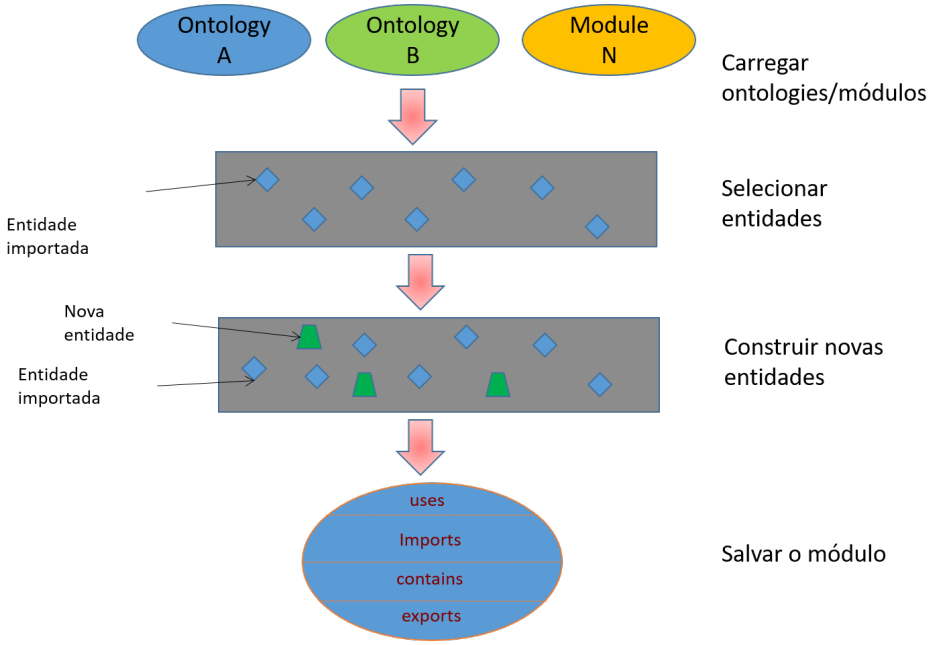


Figura 14 – Funcionamento da ferramenta Modonto

2. Checagem sintática, que assegura que o módulo não contém erros sintáticos;

3. Suporte para a utilização de alinhamentos entre os módulos importados (ou ontologias importadas) e o novo módulo;

4. Conformidade com formato de módulo padrão do projeto NEON ((HAASE et al., 2008)).

4.3.4 Hys

No artigo (MARTIN-RECUERDA; WALTHER, 2015) apresentam HyS, uma aplicação em Java que suporta uma decomposição atômica mais rápida e extração de módulos de ontologias OWL-EL. Como a quantidade de todos os possíveis módulos de uma ontologia pode ser exponencial, considerando o número de termos e axiomas de uma ontologia ((Cuenca Grau et al., 2008)); a decomposição atômica surge como uma solução para representação de módulos em tempo polinomial. HyS computa um hipergrafo direcionado ((GUEDES, 2001)) para representar a estrutura modular de uma dada ontologia. Nessa representação, um módulo baseado em localidade é equivalente a um componente composto de átomos fortemente conectados.

4.3.4.1 Localidade

Essa é uma ferramenta ((MARTIN-RECUERDA; WALTHER, 2015)) que se baseia na abordagem de extração de módulos fundamentada no conceito de localidade. Extensão conservativa pode ser utilizada como condição suficiente para noção de módulo, como foi visto

na seção 4.1.1. No entanto, desenvolver um algoritmo que extrai módulos nessa condição é altamente indecidível ((MARTIN-RECUERDA; WALTHER, 2015)). Por causa disso, surgiu o conceito de localidade, que permite fazer algoritmos decidíveis para extrair módulos.

Definição 1 1 *(**Localidade** ((Cuenca G rau e t a l., 2008))). S eja **S** u ma a ssinatura. Nós dizemos que um axioma α é local considerando S, se toda expansão trivial de uma S-interpretação para $\boldsymbol{S} \cup Sig(\alpha)$ é um modelo de α. Nós denotamos por local(S) a coleção de todos os axiomas que são locais considerando **S**. Uma ontologia **O** é local considerando **S** se $\boldsymbol{O} \subseteq$ local(S).*

Podemos admitir que uma ontologia **O** é local considerando uma assinatura **S** se nós pudermos, para qualquer interpretação dos símbolos em **S**, estendê-la para um modelo de **O**, o qual interpreta os símbolos adicionais como um conjunto vazio.

Definição 1 2 *(**Módulos b aseados n a c ondição d e l ocalidade** ((BAO e t a l., 2009))) Dada uma ontologia **Q** e uma assinatura **S**, nós podemos dizer que $Q_1 \subseteq Q$ é um S-modulo baseado em localidade em **Q** se $Q \setminus Q_1$ é local considerando $S \cup Sig(Q_1)$.*

4.3.4.2 Implementação do HyS

Os autores propuseram as seguintes definições:

Definição 1 3 *(**Módulo b aseado e m l ocalidade).** $M od^x_o(\sum)$ um módulo qu e é x-local de uma ontologia O com relação a $\sum$, o nde $x \in \{\top, \bot\}$.*

Definição 1 4 *(**Assinatura m ínima e n ão l ocal d e u m a xioma)**. C onsiste e m um subconjunto de assinaturas de axiomas tal que um axioma não é local com relação a esse subconjunto. O conjunto de todas as assinaturas de um axioma α não local e mínimo é representado por $MLS^x(\alpha)$, onde $x \in \{\bot, \top\}$.*

Definição 1 5 *(**Átomo)**. É u m c onjunto m áximo d e a xiomas f ortemente relacionados de uma ontologia no sentido que eles sempre co-ocorrem em módulos. Todos os axiomas de tais ontologias estão distribuídos sobre átomos de maneira que todo axioma ocorre em exatamente um átomo.*

HyS representa a estrutura de um ontologia O utilizando hipergrafos direcionados $H^x_O = (V, E)$ denominado hipergrafo de axiomas dependentes(*Axiom Dependency Hypergraph*) ADH, onde V é um conjunto de nós, E é um conjunto de hiper arestas e $x \in \{\top, \bot\}$. Os nós no ADH correspondem a axiomas de O e cada hiper aresta direcionada conecta um ou mais nós (calda da hiper aresta) com somente um nó (cabeça da hiper aresta). Dessa forma, a calda de uma hiper aresta direcionada τ representa o conjunto mínimo de axiomas que provê a assinatura dos símbolos requeridos por um axioma representado pelo nó cabeça de τ ser não local ((MARTIN-RECUERDA; WALTHER, 2015)).

Segundo (MARTIN-RECUERDA; WALTHER, 2015) “Módulos baseados em localidade correspondem a componentes conectados em um hipergrafo de dependência e tais componentes fortemente conectados a átomos”. Além disso, cada componente pode ser fundido com outro componente.

Eles utilizam o algoritmo de Tarjan ((TARJAN, 1972)) para computar em tempo linear todos os componentes fortemente conectados do respectivo ADH.

O HyS provê tanto uma API com conjunto de funções, como é possível utilizá-lo por linha de comando.

4.3.5 LocalityModuleExtractor de Manchester

Entre as noções mais antigas disponíveis sobre módulo, ou seja, as noções de um módulo extraído depois que a ontologia foi construída, eles utilizaram a noção de módulos baseados em localização, porque combinam fortes garantias lógicas com uma computabilidade eficiente.

Um módulo baseado em localização **M** é um subconjunto dos axiomas em uma ontologia **O** e é extraído de **O** para um conjunto **S** de termos (nomes de classe ou propriedade). O conjunto **S** é chamado de assinatura de sementes de **M**.[1]

A ferramenta está disponível tanto na Web, quanto em desktop, além de oferecer essa funcionalidade de modularização dentro da Manchester OWL API[2]. No entanto, o link disponibilizado para a ferramenta web está indisponível.

A ferramenta usa um arquivo contendo a assinatura (nomes de termos) que existe na ontologia, com os termos separados por vírgulas, espaços vazios ou barras verticais, e extrai um módulo da ontologia fornecida para o arquivo de saída especificado.

4.4 DISCUSSÃO

Na área de engenharia de ontologias, um avanço fundamental foi a proposta de propriedades formais para modularização de ontologias por (KONEV et al., 2009). Os autores chegaram a definições importantes, como a de inseparabilidade e conservatividade, além do conceito de independência de um módulo.

Algumas das abordagens apresentadas foram descontinuadas, como a $\mathcal{E}$-*connections*, que propõe a conexão entre ontologias distribuídas pela web. Por outro lado, a questão de consistência lógica entre módulos é largamente negligenciada, assim como no trabalho de PDL, que verifica consistência lógica, porém não chega ao nível de checagem lógica entre as questões de competência, que também são ignoradas nos trabalhos de modularização de ontologias.

[1] http://owl.cs.manchester.ac.uk/research/modularity/
[2] http://owl.cs.manchester.ac.uk/

Observe-se que, no caso das inconsistências advierem das questões de consistência, mesmo que uma abordagem detecte as inconsistências e estas sejam corrigidas, gera-se um novo problema: os módulos não correspondem mais às questões de consistência (enquanto requisitos, i.e., algumas não são mais atendidas pelos módulos ou pela ontologia). Além do mais, tanto a documentação da ontologia (na parte das questões de competência), quanto o rastreamento delas estão comprometidos.

Todas as abordagens apresentadas, de fato, levam em consideração uma ontologia em fase de desenvolvimento ou pronta. No entanto, devemos admitir o fato de que quando reusamos um módulo ou vários para compor uma ontologia, por exemplo, estes módulos devem ter uma especificação, que são as QCs, e elas devem ser verificadas, ou seja, deve-se verificar se o módulo está cumprindo o que foi especificado nas QCs.

Se não levar em consideração a especificação de QCs nem a sua checagem, pode haver inconsistências nesses módulos, derivadas da especificação das QCs inconsistentes. Além disso, ao juntar vários módulos, o problema se agrava, pois fica mais difícil saber realmente onde estão as inconsistências e corrigi-las. Note-se que, na abordagem proposta nesta tese, é possível identificar claramente que questões de competência entram em conflito. Nenhum dos trabalhos apresentados aborda questões de competência.

Outro ponto a ser considerado é o fato de que seria útil usar uma abordagem de modularização no início do ciclo de desenvolvimento de ontologias, ou seja, durante a fase de especificação das questões de competência. Ademais, se admitirmos uma grande ontologia, com dezenas de questões de competência, seria útil uma abordagem que modularizasse essas QCs, permitindo inclusive que elas sejam reutilizadas em outros projetos, garantindo a completude da satisfação de QCs nos módulos e das ontologias compostas por estes módulos. Além disso, poupar-se-iam tempo e custos com manutenção. Na modularização, cada QC é transformada, junto com sua resposta, num módulo que consiste em um conjunto de axiomas terminológicos.

A última versão de Modonto[3] foi disponibilizada em 2009. Em relação à ferramenta PATO, a última versão disponibilizada no site é de 2008. Apenas o HyS e a API para modularização de Manchester são abordagens mais recentes e disponíveis com o objetivo de extração de módulos. Percebemos que o Hys utiliza modularização baseada em localidade. Além dessas duas, existe uma outra ferramenta de extração de módulos baseada na noção de localidade denominada AMEX, desenvolvida por (GATENS; KONEV; WOLTER, 2014).

São as deficiências apresentadas nos parágrafos anteriores que este trabalho propõe solucionar com a construção de uma abordagem para modularização, a partir da satisfatibilidade em questões de competências escritas em linguagem natural controlada, realizando, não apenas, extração de módulos, mas também, composição e decomposição de módulos. Tal proposta se encontra no próximo capítulo.

[3] https://sourceforge.net/projects/ontocompo/

5 QUESTÕES DE COMPETÊNCIA: ESTADO DA ARTE

Neste capítulo são apresentados os fundamentos e trabalhos envolvendo questões de competência encontrados na literatura até o final da escrita desse trabalho, além de uma discussão sobre o que já existe desse assunto para solucionar problemas dentro da Engenharia de Ontologias com o uso de questões de competência.

5.1 INTRODUÇÃO

QCs foram primeiramente citadas na literatura por (FOX; GRÜNINGER, 1994) que propuseram o uso de ontologias para o contexto de engenharia de empreendimentos. Eles utilizaram questões de competência para caracterizar uma ontologia. Para isso, a ontologia deveria conter um conjunto de axiomas suficientes e necessários para responder a tais questões. As QCs eram escritas em linguagem natural.

Já que ontologias têm como objetivo principal descrever um domínio de discurso com adequado engajamento ontológico, um modo de avaliar se a modelagem desse domínio é precisa e rigorosa o suficiente para alcançar os objetivos da criação da ontologia, pode ser feito checando um conjunto de questões de competência sobre a ontologia construída. Essa checagem pode ser realizada manualmente ou por meio de deduções, extraídas do conhecimento contido na ontologia.

Resumidamente, podemos dizer que questões de competência consistem em um conjunto de questões e suas respectivas respostas, em linguagem natural, que a ontologia deve ser capaz de responder corretamente.

5.1.1 Definição Formal

Apesar de estarmos lidando com ontologias escritas em linguagem LD, nós preferimos definir o que é uma questão de competência de maneira formal e independente de determinados formalismos e linguagens. Nós apenas assumimos que existem linguagens formais para expressar ontologias e consultas. Essas linguagens e formalismos podem permitir consultas limitadas a questões/respostas, onde existe uma variável tupla que pode estar vazia.

O processo de formalização de uma ontologia define vocabulários e consequências lógicas. Abaixo, nossas definições:

Definição 1 6 ***(O ntology V ocabulary)****. Voc(O) é o vocabulário utilizado na ontologia O, isto é, a tripla* (N_C, N_R, N_O) *((BAADER et al., 2010)):*

- N_C *é um conjunto de nomes de conceitos ou conceitos atômicos(símbolos de predicado unário);*

- *N_R é um conjunto de relações ou nomes de propriedades (símbolos de predicado binário);*

- *N_O é um conjunto de nomes de indivíduos, instâncias de N_C a N_R.*

Nós denotamos por $O \models \delta$ o fato que os axiomas δ são verdadeiros em todos os possíveis modelos de O.

Nós primeiramente consideramos uma questão de competência como um par composto de uma questão e a resposta dela.

Definição 17 ***(Questões de Competência)***. *Uma questão de competência é um par $\langle Q^{NL}, \delta^{NL} \rangle$, considerando que NL significa* Natural Language*, tal que Q^{NL} é uma consulta expressa em uma linguagem e δ^{NL} é uma resposta para esta consulta expressa como uma variável de substituição.*

Definição 18 ***(Questão de competência satisfeita)***. *Uma questão de competência $\langle Q, \sigma \rangle$ é satisfeita por uma ontologia O se $O \models \sigma(Q)$.*

Na próxima seção decorremos sobre métodos de engenharia de ontologias que propõe o uso de QCs.

5.2 MÉTODOS DE ENGENHARIA DE ONTOLOGIAS BASEADO EM QUESTÕES DE COMPETÊNCIA

Posteriormente, (GRÜNINGER; FOX, 1995) propuseram uma metodologia para modelagem e avaliação de ontologias, onde sugeriram uma formalização lógica das questões de competência. Essa metodologia apresenta uma arquitetura exibida na Figura 15.

Segundo os autores, dado um cenário motivador, surgem questões de competência informais escritas em linguagem natural. A segunda etapa consiste em definir formalmente a terminologia da ontologia em termos de classes, relações e instâncias. Após isso, as questões de competência e suas respectivas respostas devem ser formalizadas utilizando predicados de primeira ordem. A quarta etapa consiste em criar axiomas sobre a terminologia. Na última fase é avaliada a competência da ontologia, através de teoremas de completude propostos pelos autores no artigo, utilizando as questões de competência.

Essas questões são geralmente escritas em linguagem natural controlada e proveem suporte ao processo de desenvolvimento de ontologias em duas principais formas:

(1) Possibilitando a identificação de elementos que fazem parte do vocabulário da ontologia como classes, instâncias e suas relações; e

(2) Provendo um meio de verificar a satisfatibilidade dos requisitos, por meio de extração de conhecimento e raciocínio sobre os axiomas da ontologia.

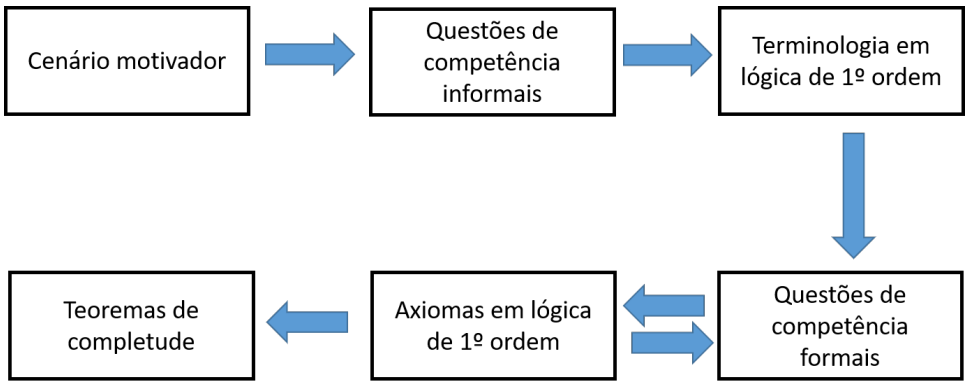


Figura 15 – Metodologia de (GRÜNINGER; FOX, 1995)

Segundo (BEZERRA; FREITAS; SANTANA, 2013), questões de competência desempenham um importante papel em todo ciclo de desenvolvimento de ontologias, já que elas representam os requisitos funcionais da ontologia.

Por exemplo, (NOY; HAFNER, 1997) sugerem o seguinte conjunto de questões de competência para o domínio de vinhos:

1. Quais características de um vinho eu devo considerar quando escolho um vinho?
   Resposta: sabor e aroma

2. Bordeaux é um vinho branco ou tinto?
   Resposta: tinto

3. Qual é a melhor escolha de vinho para uma carne grelhada?
   Resposta: Bordeaux

4. O aroma e o corpo do vinho mudam com o ano da colheita?
   Resposta: sim

Com essas questões de competência, podemos identificar o vocabulário inicial da ontologia como as classes “vinho” e “Bordeaux”, e a relação “ano da colheita”. Além disso, após a construção da ontologia, podemos verificar a competência da ontologia em respondê-las.

Nas metodologias de ontologias, QCs são mencionadas apenas na fase inicial e essas metodologias sugerem o uso de linguagem natural para sua construção, utilizando geralmente aplicativos de texto.

Nas próximas seções apresentaremos alguns trabalhos que fazem uso de QCs em fases de desenvolvimento de ontologias, como a área de especificação do escopo da ontologia, a área de testes e avaliação de ontologias, a área de autoria de ontologias, e as áreas de modelagem e rastreabilidade de ontologias.

5.3 QUESTÕES DE COMPETÊNCIA COMO ESPECIFICAÇÃO DE REQUISITOS DE ONTOLOGIAS

Uma das fases mais importantes do processo de desenvolvimento de software é a especi-ficação de requisitos. Especificações de requisitos de software consistem na descrição das funcionalidades desejáveis de um sistema.

(HOFMANN; LEHNER, 2001) afirmam que encontrar e corrigir um problema de software após a entrega do sistema ao cliente é 100 vezes mais caro do que durante os requisitos e as fases iniciais do projeto. Isso significa que contradições, negligência e ambiguidade não descobertas anteriormente, vão trazer consequências indesejáveis.

Essa declaração também se aplica à engenharia de ontologias. Uma ontologia mal especificada, p rovavelmente, n ão v ai c umprir o s o bjetivos d esejados d o e ngenheiro de ontologias e do cliente. Além disso, pode provocar excesso de trabalho e, assim, custos de manutenção mais elevados.

O (IEEE, 1993) afirma q ue o d ocumento d e e specificação de re quisitos de software deve ser "correto, não ambíguo, completo, consistente, listado por importância e/ou por estabilidade, verificável, modificável e rastreável". Tais aspectos também são desejados na fase de especificação de requisitos de ontologias.

(SUÁREZ-FIGUEROA et al., 2008) propuseram guias metodológicos para a construção do documento de especificação de requisitos de ontologias (ORSD), onde os autores se fundamentaram nos componentes *Por quê? O quê? Quem? Onde? Quando? Como?* e em questões de competência para orientar a criação do ORSD. Todos esses guias são descritos utilizando formulários e *workflows*. O *workflow* que exibe a proposta dos autores é exibido na Figura 16.

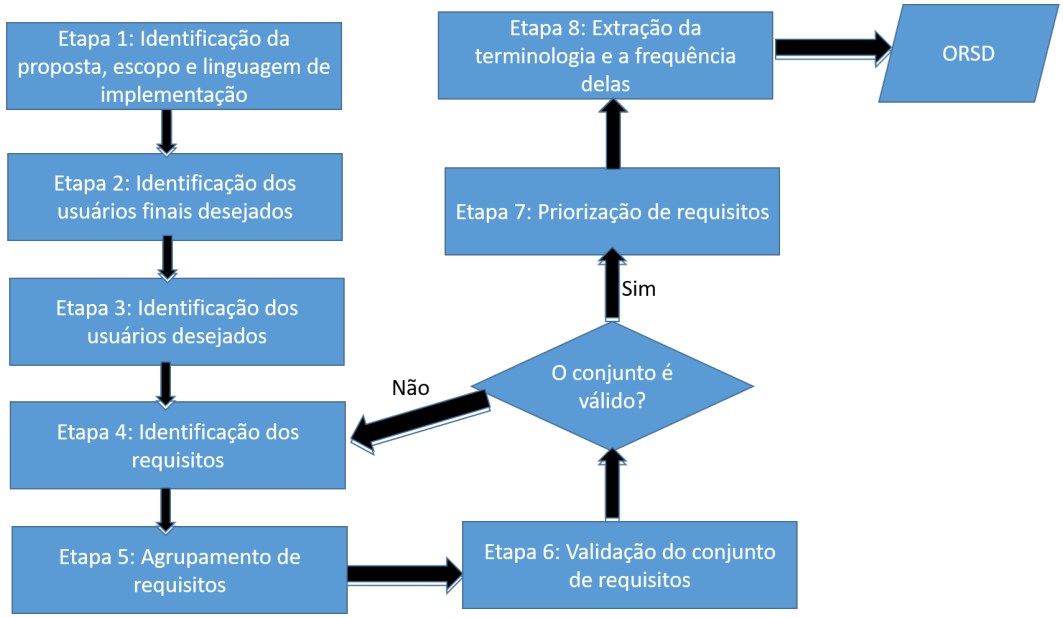


Figura 16 – Workflow de (SUÁREZ-FIGUEROA et al., 2008)

Executando o *workflow*, conseguimos identificar as respostas para as perguntas *Por quê? O quê? Quem? Onde? Quando? Como?* nas quais a proposta se baseia, como por exemplo, a identificação dos usuários finais e dos especialistas em domínios que respondem à pergunta *Quem?*.

Na etapa 4, são identificados os requisitos descritos como questões de competência. Após isso, as QCs são agrupadas em categorias e, na próxima etapa, o conjunto de requisitos é validado. Os critérios de validação foram inspirados por (DAVIS, 1993) e (IEEE, 1993).

Segundo os autores, o principal objetivo do ORSD "é servir como um acordo entre engenheiros de ontologias, usuários e especialistas de domínio sobre quais requisitos a ontologia deveria cobri". O uso de ORSD traz benefícios para o desenvolvimento de software((SUÁREZ-FIGUEROA; GÓMEZ-PÉREZ; VILLAZÓN-TERRAZAS, 2009)), como:

1. Identificação de qual conhecimento particular deveria ser representado na ontologia;
2. Facilitar o reuso de recursos de conhecimento existentes;
3. Facilitar a verificação da ontologia, no que diz respeito às exigências que a ontologia deve cumprir.

Na próxima seção, apresentaremos trabalhos sobre questões de competência para teste de ontologias utilizando *unit test*.

5.4 QUESTÕES DE COMPETÊNCIA PARA TESTAR ONTOLOGIAS QUE UTILIZAM A ABORDAGEM DE UNIT TESTS

Teste de unidades (*Unit Testing*) em Engenharia de Software ((RUNESON, 2006)), consiste na criação de pequenos testes, geralmente pequenas unidades de código, as quais podem ser avaliadas de maneira rotineira por desenvolvedores para checar possíveis erros, principalmente após mudanças no software. Nesse método de teste, também é importante um procedimento para a correção do erro (chamado de bug), o qual consiste na criação de um teste que provê evidência para o bug antes de corrigi-lo.

5.4.1 Unit tests para Ontologias

(VRANDEČIĆ; GANGEMI, 2006) dão várias motivações para o uso de testes de unidade (*unit testing*) da engenharia de software que são partes pequenas de software utilizadas para testar mais facilmente o sistema.

Os autores investigaram os benefícios de testes unitários para ontologias:

1. para facilitar testes de regressão;
2. prover um framework de teste que possa crescer incrementalmente durante a fase de manutenção e evolução; e ainda

3. ser simples de utilizar.

No artigo, eles apresentam algumas abordagens que podem ser utilizadas como a ideia de modelo por contrato (*design by contract*), na qual se estabelecem quais declarações deveriam ou não ser derivadas de ontologia desenvolvida ou mantida, assim como declarações explícitas ou questões de competências, utilizando uma linguagem de consulta.

Além dessa abordagem, eles também sugerem utilizar lógica autoepistêmica proposta por (MOORE, 1985), ccomo uma boa maneira de formalizar a introspecção de ontologias ((VRANDEČIĆ; GANGEMI, 2006)).

5.4.2 OntologyTest Tool

Neste artigo, proposto por (GARCÍA-RAMOS; OTERO; FERNÁNDEZ-LÓPEZ, 2009), foi proposto um método e uma ferramenta, denominada OntologyTest, para testar requisitos funcionais de uma ontologia em OWL-DL. A ferramenta permite que os próprios usuários definam os testes, que equivalem a um conjunto de instâncias, uma consulta e um resultado esperado. O usuário executa os testes e depois pode inspecionar os resultados da execução.

Segundo os autores, a ferramenta pode ser vista como o complemento da metodologia de (GRÜNINGER; FOX, 1995), e a abordagem de desenvolvimento orientado a questões de competência proposta pelo projeto Neon.

A proposta dos autores permite que um engenheiro de ontologias, utilizando a OntologyTest, capture os requisitos funcionais da ontologia e verifique tais requisitos durante o ciclo de desenvolvimento de ontologias, ajudando a detectar erros durante as fases de desenvolvimento e validação. A execução consiste nos seguintes passos:

- O usuário cria um projeto na ferramenta;
- Define qual ontologia será testada. A ontologia deve ser consistente e sintaticamente correta;
- Define as consultas na linguagem SPARQL ((ANGLES; GUTIERREZ, 2008)) que devem ser satisfeitas e um conjunto de instâncias;
- E, para cada consulta, especificar qual o resultado esperado.

A OntologyTest suporta os seguintes testes:

- **Teste de instanciação:** testa se um objeto pertence ou não a uma classe;
- **Teste de recuperação:** onde é possível recuperar uma lista de instâncias de uma classe;
- **Teste de realização:** estabelece a classe mais específica que deve ser instanciada por uma instância;

- **Teste de satisfação:** especifica se uma inconsistência poderia surgir na ontologia depois de adicionar uma nova instância;
- **Teste de classificação:** especifica uma lista com todas as classes às quais uma instância deve pertencer;
- **Teste SPARQL:** as consultas são escritas em SPARQL ((ANGLES; GUTIERREZ, 2008)) e os resultados obtidos são comparados com os resultados esperados.

Os autores afirmaram que a ferramenta proposta foi avaliada com ontologias de médio e grande portes, e foram feito testes em condições de estresse, porém, não é dito quais ontologias utilizadas, quantidade de classes, propriedades e axiomas, quanto tempo levou para executar os testes, em quais testes a ferramenta falhou, nem suas limitações.

5.4.3 Metodologia e ferramenta para teste de ontologias proposto por (BLOMQVIST; SEPOUR; PRESUTTI, 2012)

Neste trabalho ((BLOMQVIST; SEPOUR; PRESUTTI, 2012)), os autores propuseram uma metodologia e uma ferramenta para testar requisitos funcionais de ontologias, alegando que não existia, até então, uma metodologia detalhada e fundamentada em unit testing da engenharia de software. Os autores, em sua metodologia, também propõem o uso de questões de competência. A ferramenta foi implementada como parte do *XD Tools* ((PRESUTTI et al., 2009)).

Eles estabeleceram o conceito de *Test Case* no âmbito da engenharia de ontologias, como um repositório que armazena informação sobre um teste, que é criado para cada requisito da ontologia. Formalmente, eles representaram um Test Case como uma ontologia OWL, identificada por uma Uniform Resource Indicator (URI), normalmente sendo um ABox que depende do TBox da ontologia que será testada.

Eles criaram um metamodelo (ver Figura 17) de teste em OWL, para viabilizar os testes na prática. No metamodelo é ilustrado que um *Test Case* é uma instância de uma ontologia, que tem relação com um requisito e um procedimento de teste. Um *Test Case* é definido como um membro de um *Test Suite*, que é uma coleção de *Test Cases*. Um *Test Case Run* é a execução de um *Test Case* sobre uma versão específica da ontologia. O metamodelo ainda provê armazenamento de todas as execuções dos testes.

A metodologia apresenta os seguintes passos:

1. **Coletar requisitos:** Para um específico tipo de teste, obter todos os requisitos do módulo atual que são relevantes para este tipo de teste;
2. **Seleção de requisitos:** Seguindo o princípio de *unit testing*, selecionar um ou mais requisitos para o teste em cada *test case*;
3. **Procedimento de teste:** Determinar como testar um particular requisito;

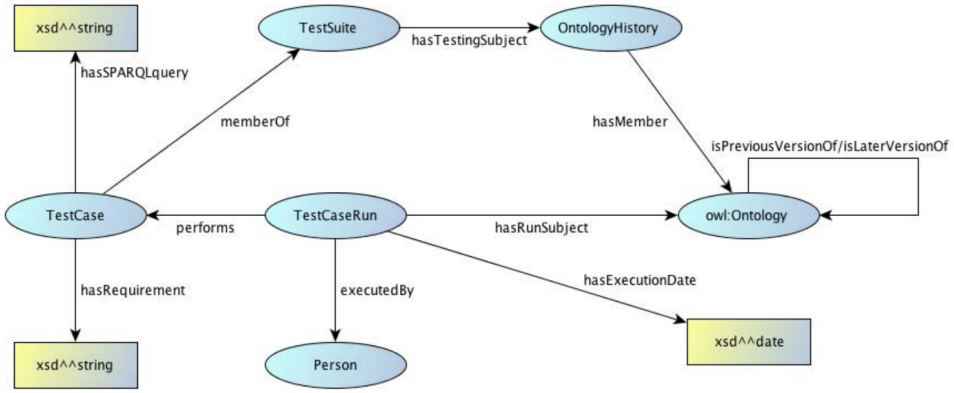


Figura 17 – Partes do TestCase meta model em OWL. Fonte: (BLOMQVIST; SEPOUR; PRESUTTI, 2012)

4. **Criação de *test case*:** Criação de um arquivo OWL com o *test case* e um arquivo OWL adicional para incluir o primeiro *test run* e descrever ambos utilizando um metamodelo de test case e suas propriedades;

5. **Adição de dados do teste:** Adicionar dados do teste necessários para realizar o passo três, em um arquivo OWL *test run* (ou em um módulo de dados separado);

6. **Determinação de resultados esperados:** Dependendo do teste de dados, qual seria a saída *negative test run*, isto é, onde os requisitos são cumpridos?

7. ***Run Test*:** Executar o passo três sobre o arquivo OWL *test run* com os dados do passo cinco e salvar resultados;

8. **Comparar resultados:** Verificar a saída esperada (passo 6) com resultado real da etapa sete;

9. **Análise resultados inesperados:** Se o resultado não é esperado, isto é, um resultado positivo, analisar o motivo e documentar sugestões de mudanças ou questões;

10. **Documentação:** Armazenar todas as informações sobre o *test run* e a relação dele com os seus *test case*, utilizando propriedades do metamodelo de teste;

11. **Repetir:** Se existirem mais requisitos para testar, retornar ao passo dois.

E ela é composta basicamente de três tipos de teste:

- **Verificação de QC:** a verificação é feita utilizando consultas Semantic Query Language for Databases (SPARQL), comparando o resultado obtido com o esperado;
- **Verificação de Inferência:** é utilizado um raciocinador para fazer classificação de indivíduos;

- **Provocação de erro:** o sistema recebe dados incorretos para checar se o sistema irá realmente classificá-los como incorretos. Um modo de fazer isso em ontologias é checando a consistência. Por exemplo, considerando que uma classe pessoa não pode ser homem e mulher ao mesmo tempo, o sistema é alimentado com instâncias de ambas as classes. No caso, o correto seria o sistema detectar uma inconsistência.

A ferramenta também sugere consultas SPARQL através de uma análise linguística da QC dada em linguagem natural. Na avaliação, foi utilizada a métrica de taxa de erro em ontologias disponíveis nos repositórios citados no (BLOMQVIST; SEPOUR; PRESUTTI, 2012).

Na próxima seção, apresentaremos trabalhos que utilizam QCs na etapa de autoria de ontologias.

5.5 QUESTÕES DE COMPETÊNCIA NA AUTORIA DE ONTOLOGIAS

5.5.1 Trabalho de (REN et al., 2014)

(REN et al., 2014) afirmam que autoria (criação) de ontologias é uma tarefa difícil para quem não é especialista em lógica, tanto no processo de criação dos requisitos, como testar se os requisitos satisfazem a ontologia. O objetivo do artigo é tentar resolver esse problema com uma nova abordagem que leva em consideração questões de competência e o desenvolvimento de um software. Para isso, eles realizaram um estudo de pesquisa no qual tentam encontrar respostas para as seguintes perguntas:

1. Como as QC são formuladas no mundo real?; e
2. Como podemos automatizar teste se uma QC pode ser respondida de forma significativa?.

Para responder à primeira pergunta, eles fizeram um estudo com QCs do mundo real de diferentes domínios, criados por diferentes autores com níveis de especialidade diferente. Com esse estudo, eles categorizaram QCs em padrões que diferenciam um do outro por características. Para responder à segunda pergunta, eles identificaram pressupostos de questões de competência associadas a uma ontologia. Por pressuposto, eles entenderam que é um termo que se refere a uma condição especial que deve ser satisfeita para uma expressão linguística ter uma denotação. Por exemplo, “Quais as coberturas de uma pizza portuguesa?” parte do pressuposto de que existe uma pizza portuguesa na ontologia.

Eles consideraram QCs como perguntas em linguagem natural e, para encontrar os padrões da primeira pergunta, eles utilizaram SPARQL para transformar a QC em linguagem natural para uma consulta SPARQL. Os padrões encontrados depois do estudo são exibidos na Figura 18. Para definição dos padrões os autores criaram *archetypes*, que correspondem a especificação dos pedaços da QC:

- **PA:** *Predicate Arity*
- **RT:** *Relation Type*
- **M:** *Modifier*
- **DE:** *Domain-independent Element*
- **OBJ.:** *Object property relation*
- **DATA.:** *Datatype property relation*
- **NUM.:** *Numeric modifier*
- **QUAN.:** *Quantitative modifier*
- **TEM.:** *Temporal element*
- **SPA.:** *Spatial element*
- **CE:** *Class expression*
- **OPE:** *Object property expression*
- **DP:** *Datatype property*
- **I:** *Individual*
- **NM:** *Numeric modifier*
- **PE:** *Property experssion*
- **QM:** *Quantity modifier*

Baseado nas categorias e características encontradas na primeira etapa do trabalho, eles analisam e definem tipos de testes para responder cada QC. Os autores definiram testes de checagem de satisfatibilidade de classes, de relações, de cardinalidades, dente outros apresentados no artigo (REN et al., 2014).

Por exemplo, para QC "*What is the best software to read this data*?" pertence ao padrão 7: What [CE1] is [NM] to [OPE][CE2]?. A partir do QC e seu padrão, o sistema pode extrair automaticamente o elementos da QC. Nesse caso, é uma questão de seleção do tipo "*What*" contendo um predicado de três aridades (entre "*software*", "*data*" e "*read*") com um valor numérico modificador ("*best*"), que deve ser modelado como uma classe e algum objeto implícito e propriedades de tipo de dados, cujos nomes podem ser gerados a partir de contextos ou atribuídos por usuários. Em seguida, o sistema gera automaticamente testes, denominados ATs, para checagem de satisfatibilidade de classes, de relações, de cardinalidades, dente outros.

ID	Padrão	Exemplo	PA	RT	M	DE
1	Which [CE1][OPE][CE2]?	Which pizzas contain pork?	2	Obj.		
2	How much does [CE][DP]?	How much does Marguerita Pizza weigh?	2	Data.		
3	What type of [CE] is [I]?	What type of software is it?	1			
4	Is the [CE1] [CE2]?	Is the software open source development?	2			
5	What [CE] has the [NM] [DP]?	What pizza has the lowest price?	2	Data.	Num.	
6	What is the [NM] [CE1] to [OPE] [CE2]?	What is the best/fastest/most robust software to read/edit this data?	3	Both.	Num.	
7	Where do I [OPE] [CE]?	Where do I get updates?	2	Obj.		Spa.
8	Which are [CE]?	Which are gluten free bases?	1			
9	When did/was [CE] [PE]?	When wasthe 1.0 version released?	2	Data.		Tem.
10	What [CE1] do I need to [OPE] [CE2]?	What hardware do I need to run this software?	3	Obj.		
11	Which [CE1] [OPE] [QM] [CE2]?	Which pizza has the most toppings?	2	Obj.	Quant.	
12	Do [CE1] have [QM] values of [DP]?	Do pizzas have diferente values os size?	2	Data.	Quant.	

Figura 18 – Padrões encontrados pela abordagem Question-driven Ontology Authoring

A continuidade desse trabalho é mostrado em (DENNIS et al., 2017). Os autores propuseram autoria de ontologias baseada em QCs(CQOA), ferramenta com uma interface que checa continuamente, durante a autoria/desenvolvimento da ontologia quais QCs estão satisfeitas de acordo com uma dada ontologia. Na ferramenta também é possível interagir através de comandos em linguagem natural controlada.

Os autores trazem o conceito de pressuposição aplicada à QC, que consiste em, dada uma QC, uma proposição que precisa ser verdadeira para que a pergunta tenha uma resposta. Cada tipo de pressuposto é mapeado para algum teste de autoria (AT); cada um é testável usando serviços de raciocínio ou verificação de subsunção na linguagem ALC DL. Por exemplo, para "Quais pizzas contêm queijo?" gera dois presupostos:
Positivo: Pizza $\sqcap \exists$contém.Queijo *é satisfatível*
Negativo: Pizza $\sqcap \exists$contém.Queijo *não é satisfatível*

Segundo (DENNIS et al., 2017)

> "CQOA é potencialmente poderoso porque pode ajudar os autores de ontologias a entenderem, a cada estágio do processo de criação, se QCs que eles especificaram foram criadas corretamente pela ontologia que eles estão construindo".

Eles fizeram experimentos com três grupos de locais diferentes. Foi dado aos participantes uma ontologia simples, criada pelos autores, e sete questões de competência. Seguindo isto, os participantes foram adicionando as sete QCs, uma por uma. Para cada QC, a ferramenta mostrava aos participantes os ATs associados da QC e foi perguntado aos participantes se em cada caso eles concordavam com as ATs geradas.

Os resultados sugeriram que o apoio para muitos pressupostos está longe de ser completo, onde alguns deles foram apoiados por 36% dos participantes.

5.6 QUESTÕES DE COMPETÊNCIA NA AVALIAÇÃO DA SATISFATIBILIDADE DE ONTOLOGIAS

Nesta seção serão apresentados trabalhos com o objetivo de utilizar questões de competência para avaliação de ontologias.

5.6.1 Competency Evaluation tool

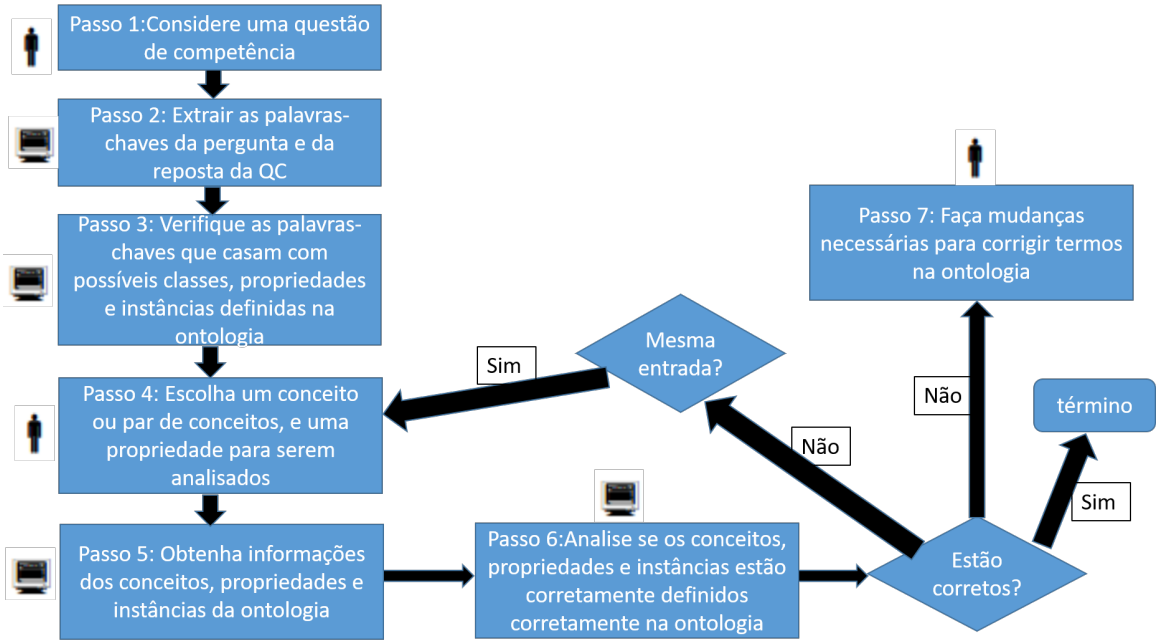


Figura 19 – Passos da abordagem de (ANNAMALAI; SANIP, 2010)

No artigo de ((ANNAMALAI; SANIP, 2010)), o objetivo é propor uma abordagem que utilize questões de competência em linguagem natural para avaliar semanticamente ontologias durante todo o desenvolvimento delas, ou seja, que a avaliação não seja realizada apenas no final do ciclo, depois da ontologia desenvolvida e sobre as instâncias. Um outra contribuição é uma ferramenta que implementa essa abordagem e um plugin para o Protégé ((MUSEN, 2015)).

Os autores propuseram um conjunto de passos, exibidos na Figura 19. Alguns passos são automatizados, como os passos 2 e 5. Outros precisam ser realizados manualmente, como os passos 1, 4 e 6. Na tela da ferramenta, o usuário entrou com a QC "In which news sections are advertisements allowed?", ao clicar no botão "find terms", a ferramenta extrai possíveis palavras-chaves da QC e também uma lista de conceitos, relações e propriedades da ontologia. O usuário pode ainda analisar melhor conceitos e propriedades da ontologia no trecho "Retrieve background information about".

A performance da ferramenta, geralmente relacionada com tempo e eficiência, foi avaliada de duas maneiras ((ANNAMALAI; SANIP, 2010)):

- Verificar a habilidade da ferramenta de avaliar QCs;
- Foi feito um teste de usabilidade com usuários em potenciais no laboratório, no intuito de observar o quão bem a ferramenta os auxiliava nesse trabalho de avaliação de ontologias.

5.6.2 Abordagem semi-automática de avaliação de ontologias biomédicas para o domínio de biobanking baseado em questões de competência

(HOFER et al., 2015) propuseram uma abordagem semiautomática de avaliação que permite identificar ontologias relevantes para o domínio do *biobanking* baseada em questões de competência. *Biobanks* armazenam amostras biológicas juntamente com dados clínicos, consentimento informado em declarações, condições de processamento e armazenamento. A abordagem consiste em três estágios:

1. **Identificação das ontologias candidatas:** eles utilizaram diversos portais de ontologias, como *National Center for Biomedical Ontology*[1], para seleção das ontologias. Para fornecer uma prova conceitual da abordagem, foi definido usar OMIABIS ((BROCHHAUSEN et al., 2013))como a ontologia de referência. OMIABIS é baseado nos dados do MIABIS ((NORLIN et al., 2012)),modelo que consiste em uma representação em padrão internacional do conteúdo de um *biobanking*;
2. **Identificação de questões de competência:** foram criadas questões de competência de forma manual para as ontologias selecionadas;
3. **Abordagem de avaliação:** Avaliar as ontologias utilizando as QCs. Para isso, eles criaram um algoritmo que recebe uma QC como entrada. Como saída, verifica se uma questão de competência é coberta por uma ontologia. Isso é verdade se cada token relevante da QC puder ser mapeado para um conceito, propriedade de objeto ou instância da ontologia. O algoritmo extrai termos, como conceitos e propriedades da ontologia, e constrói uma consulta na linguagem SPARQL.

Os próprios autores conduziram os testes. Uma limitação da nossa abordagem de avaliação é a correspondência de tokens únicos, o que leva a um maior número de correspondências falso-positivas como palavras em questões de competência.

5.7 METODOLOGIAS PARA MODELAGEM E RASTREABILIDADE DE QUESTÕES DE COMPETÊNCIA

Nesta seção são apresentados trabalhos para modelagem do escopo da ontologia utilizando questões de competência e um trabalho sobre rastreabilidade de questões de competência.

[1] http://bioportal.bioontology.org/ontologies

5.7.1 Uso da metodologia Tropos para modelar Questões de Competência

A principal contribuição desse trabalho ((FERNANDES; GUIZZARDI; GUIZZARDI, 2011)) é propor um método que utiliza modelagem baseada em objetivos para capturar e modelar questões de competência. Para isso, eles sugerem o uso de uma metodologia de engenharia de software orientada a agentes, denominada, Tropos ((BRESCIANI et al., 2004)), a qual é fundamentada nos conceitos de ator (usuários, desenvolvedores etc.) e objetivo.

- **Requisitos iniciais**: Entendimento do problema, da organização em si, antes de ser construída. A saída dessa atividade é um modelo que inclui atores, os objetivos desses atores e as relações de interdependência.
- **Requisitos finais**: Nessa fase, uma solução é proposta e revisada, ocorrendo o refinamento do modelo construído na fase anterior.

A metodologia de engenharia de ontologia proposta é definida por:

1. Nos requisitos iniciais, os objetivos de um dado ator do sistema, de uma determinada organização são analisados, seguindo os métodos de análise defendidos pela metodologia Tropos;
2. Nos requisitos finais, questões de competência são definidas para atingir os objetivos dos atores;
3. Quando uma solução é finalmente encontrada na forma de um sistema baseado em ontologia, as dependências são estabelecido entre os atores do sistema, através das questões de competências.
4. Por fim, a modelagem que pode ser feita usando diagramas ER ou UML.

Os autores realizaram um estudo de caso numa unidade de saúde para validar a abordagem proposta.

5.7.2 Um Método de Desenvolvimento de Ontologias Baseado em Questões de Competência com Rastreabilidade Automática

Nesse trabalho ((MALHEIROS; FREITAS, 2013)), é introduzido um método de desenvolvimento de ontologias semiautomático usando questões de competência. O trabalho apresenta as seguintes contribuições((MALHEIROS; FREITAS, 2013)):

- um rastreador que monitora as relações entre requisitos e código;
- um componente que traduz questões em linguagem natural para lógica de descrição para efetuar checagem automática usando raciocinadores;

- um gerador de questões de competência; e
- um componente que escreve código OWL de acordo com perguntas e respostas.

A implementação dessas contribuições é baseada na arquitetura mostrada na Figura 20. Os componentes são ((MALHEIROS; FREITAS, 2013)):

- **Verificador:** recebe uma QC escrita em linguagem natural e sua resposta esperada, para verificar se uma ontologia possui o conhecimento necessário para responder à pergunta com a resposta passada como entrada;
- **Questionador:** tem como objetivo formular perguntas, que devem ser respondidas pelo usuário, para gerar novos conhecimentos que serão adicionados à ontologia. Para isso, o componente recebe uma QC e sua resposta esperada como entrada, criando a partir delas novas perguntas que, ao serem respondidas, vão gerar o conhecimento necessário para responder à QC passada como entrada com a sua resposta esperada;
- **Construtor:** é acionado quando o usuário responde a uma pergunta gerada pelo componente Questionador. O seu objetivo é transformar uma pergunta e sua resposta em código OWL e adicioná-la em uma ontologia;
- **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.

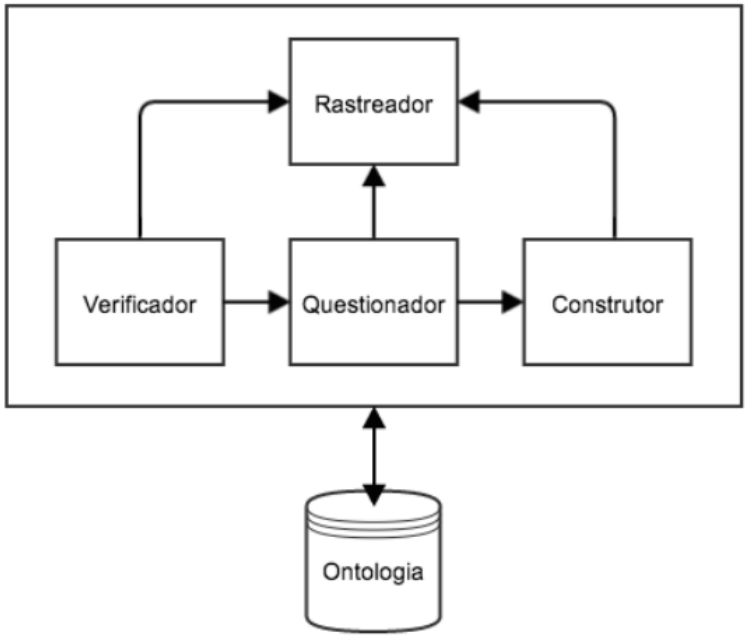


Figura 20 – Arquitetura do trabalho proposto por (MALHEIROS; FREITAS, 2013)

5.8 DISCUSSÃO

Apresentamos metodologias que propõem o uso de QCs no desenvolvimento de ontologias, bem como trabalhos que utilizam QCs em etapas do desenvolvimento de ontologias. Com relação às metodologias apresentadas que apregoam o uso de questões de competência para especificação de requisitos da ontologia, algumas delas fazem checagem manual para verificar se uma ontologia foi criada conforme as QCs (sendo estas últimas especificadas no início do desenvolvimento) como, por exemplo, o trabalho proposto por (SUÁREZ-FIGUEROA et al., 2008) no contexto do importante projeto NeOn (*NEtworked ONtologies*, que recebeu quase 15M de euros de investimento de pesquisa e desenvolvimento).

Também mostramos trabalhos que utilizam QCs para especificação dos requisitos de uma ontologia, como a proposta de (SUÁREZ-FIGUEROA et al., 2008), acima citada. É digno de nota que os requisitos de uma ontologia, i.e., as questões de competência podem ser modificadas ao longo do tempo, embora isso seja menos comum do que na área de engenharia de software. Nesse caso, uma abordagem semiautomática permitiria a checagem de consistência entre as questões, o que seria mais trabalhoso de ser realizado de forma manual.

Falando em métodos semiautomáticos de checagem de QCs, ((FERNANDES; GUIZZARDI; GUIZZARDI, 2011)) propuseram um método que utiliza modelagem baseada em objetivos, como a metodologia Tropos, para capturar e modelar questões de competência, onde o modelo era representado em diagramas UML ou ER.

Com relação ao uso de QCs para testes de ontologias, (VRANDEČIĆ; GANGEMI, 2006) propuseram ideias interessantes para o uso de questões de competência, como o uso de modelo por contrato e lógica autoepistêmica. No entanto, o artigo lida apenas com sugestões de aplicar *unit tests* em engenharia de ontologias; nada de concreto foi feito ou sugerido, nem mesmo uma metodologia. A partir desse primeiro trabalho, outras abordagens ((BLOMQVIST; SEPOUR; PRESUTTI, 2012) e (GARCÍA-RAMOS; OTERO; FERNÁNDEZ-LÓPEZ, 2009)) mais elaboradas foram propostas que, de fato, utilizavam o modelo de *unit testing*, além de ferramentas. No entanto, os trabalhos utilizam SPARQL para realizar os testes, ou seja, os testes são realizados apenas sobre o conhecimento assercional (ABox) da ontologia.

Também o uso de questões de competência na autoria de ontologias, para auxiliar os usuários, que não são especialistas em linguagens formais, a criarem e testarem ontologias surgiu no contexto de pesquisas dessa área. (REN et al., 2014) propuseram um conjunto de padrões para criação de questões de competência em linguagem natural controlada, considerando o conhecimento terminológico da ontologia (TBox) para facilitar a validação dessas QCs. No entanto, esse trabalho cita o autor da tese, que já tinha proposto padrões de QCs para checagem de consistência de QCs em (BEZERRA; FREITAS; SANTANA, 2013).

Em (DENNIS et al., 2017) foi apresentada uma ferramenta que utiliza tais padrões e

testes de autoria que são gerados a partir das QCs. O trabalho não apresenta o conceito de modularização a partir de questões de competência e checagem de consistência entre módulos. Na utilização de QCs na avaliação da satisfatibilidade de ontologias, (ANNAMALAI; SANIP, 2010) propuseram uma característica nova, que consistia em avaliar a ontologia em todo o ciclo de desenvolvimento utilizando QCs, e não apenas no final do desenvolvimento, além de uma ferramenta para este fim. Todavia, da mesma forma que as abordagens anteriores, a avaliação era feita apenas considerando o conhecimento assercional (ABox) da ontologia, utilizando SPARQL.

Um trabalho mais recente proposto por (HOFER et al., 2015), utiliza questões de competência na avaliação da satisfatibilidade de ontologias biomédicas. Os autores propõem uma abordagem semiautomática de avaliação, a qual permite identificar ontologias relevantes para o domínio do biobanking baseadas em questões de competência. Entretanto, na abordagem, eles propõem a utilização de consultas Sparql, apenas conhecimento assercional (ABox), sem a utilização de modularização como os demais trabalhos.

Por fim, existe uma pesquisa pioneira em rastreabilidade de QCs proposta por ((MALHEIROS; FREITAS, 2013)), além de uma metodologia para o desenvolvimento de ontologias.

Contudo, na revisão da literatura, não foi possível encontrar trabalhos com modularização de ontologias baseada na satisfatibilidade de questões de competência. Assim sendo, é nessa lacuna que a proposta deste trabalho se encaixa, qual seja, criar uma abordagem para modularização baseada na satisfatibilidade de QCs escritas em linguagem natural controlada; modularização de ontologias a partir da satisfatibilidade de QCs, no início do ciclo do desenvolvimento com checagem de insatisfatibilidade entre uma ou mais QCs. Isso poupará tempo dos desenvolvedores, tanto na fase de criação da ontologia, como na etapa de validação, além de promover o reuso.

Ressalte-se que, tanto a modularização, como a checagem de satisfatibilidade no início do ciclo de desenvolvimento de ontologias a partir de QCs escritas em linguagem natural controlada não tinham sido ainda propostas na literatura.

6 UMA ABORDAGEM DE MODULARIZAÇÃO DE ONTOLOGIAS BASEADA NA SATISFAÇÃO LÓGICA DE QUESTÕES DE COMPETÊNCIA

Neste capítulo, abordaremos o trabalho proposto que consiste, basicamente, em duas partes:

1. Criar uma abordagem e sua respectiva implementação para modularização de ontologias baseada em questões de competência; e
2. Verificação de inconsistência entre QCs. Não encontramos nenhum trabalho que verifique inconsistência entre QCs desde o início da engenharia de ontologias.

Vale ressaltar que a inconsistência entre módulos é pouco explorada na literatura, podendo surgir devido a questões de competência inconsistentes que não foram verificadas no início do ciclo de desenvolvimento do módulo/ontologia.

Além disso, não encontramos, na literatura, métodos ou metodologias que também considerassem a checagem de satisfatibilidade no início do desenvolvimento, através da definição d as q uestões d e competência.

Essas duas características da abordagem fazem com que o desenvolvedor descubra erros e melhorias no início do processo, economizando tempo no restante do ciclo de desenvolvimento e, com efeito, usando essa técnica o desenvolvedor, desde o início, pode ter uma ontologia modularizada que traz benefícios, como reutilização de módulos e escalabilidade.

Nos capítulos anteriores, foram apresentadas várias abordagens de modularização, mas nenhuma considera a modularização na fase inicial do ciclo de desenvolvimento de ontologias. Também vimos várias abordagens e aplicações que utilizam questões de competência, principalmente para fase de verificação d os r equisitos, p orém n enhuma c onsiderava a modularização de ontologias utilizando QCs na fase inicial.

6.1 DEFINIÇÕES FORMAIS

Abaixo, algumas definições formais:

Definição 1 9 ***(Ontology Vocabulary)****. Voc(O) é o vocabulário utilizada na ontologia O, isto é, a tripla* (N_C , N_R , N_O)*((BAADER et al., 2010)):*

- N_C *é um conjunto de nomes de conceitos ou conceitos atômicos (símbolos de predicado unário);*
- N_R *é um conjunto de relações ou nomes de propriedades (símbolos de predicado binário);*

- *N_O é um conjunto de nomes de indivíduos, instâncias de N_C a N_R.*

Nós denotamos por $O \models \delta$ o fato que os axiomas δ são verdadeiros em todos os possíveis modelos de O.

Nós primeiramente consideramos uma questão de competência como um par composto de uma questão e a resposta dela.

Definição 20 ***(Questões de Competência)***. *Uma questão de competência é um par $\langle Q^{NL}, \delta^{NL} \rangle$, considerando que NL significa* Natural Language*, tal que Q^{NL} é uma consulta expressa em uma linguagem e δ^{NL} é uma resposta para esta consulta expressa como uma variável de substituição.*

Definição 21 ***(Questão de competência satisfeita)***. *Uma questão de competência $\langle Q, \sigma \rangle$ é satisfeita por uma ontologia O se $O \models \sigma(Q)$.*

Definição 22 ***(Módulo)***. *Um módulo M é um conjunto de axiomas $\sum$ para a QC (i.e., $\langle$ Q,σ $\rangle$) tal que $\sum \cup Q \models \sigma$.*

Definição 23 ***(Módulo/Vocabulário da Questão de Competência)***. *Um módulo ou vocabulário de pergunta de competência, denotado por Voc(QC) ou Voc(M), é o subconjunto do vocabulário Voc(O) composto por nomes de classes e propriedades em $\sum$, e o conjunto de axiomas $\sum$ de M que pode responder Q. Se uma QC_i é um subconjunto de QC, então $Voc(QC_i) \subseteq Voc(QC)$, e o conjunto de axiomas é $\sum_i$ tal que $\sum \subseteq \sum_i$.*

Definição 24 ***(Satisfatibilidade de módulos)***. *Um módulo M é satisfeito se suas QCs e subQCs são satisfeitas pela ontologia O ,ou seja, $O \cup Q \models \sigma$.*

6.2 ABORDAGEM PROPOSTA

A Figura 21 apresenta a abordagem proposta.

1. O primeiro passo é criar questões de competência em linguagem natural controlada (a ser explicada na seção 6.4.4), já que estamos lidando com limitações do processamento da linguagem natural.
   ⇒ As limitações para criar uma QC consistem apenas de criar seguindo as regras da gramática (a ser explicada na seção 6.4.4).

2. Depois é realizada a modularização das QCs, considerando que uma questão de competência pode gerar novas questões para poder respondê-la. Essas questões são chamadas subQCs. A QC e suas subQCs são consideradas um módulo que pode ter conexões com outros módulos;

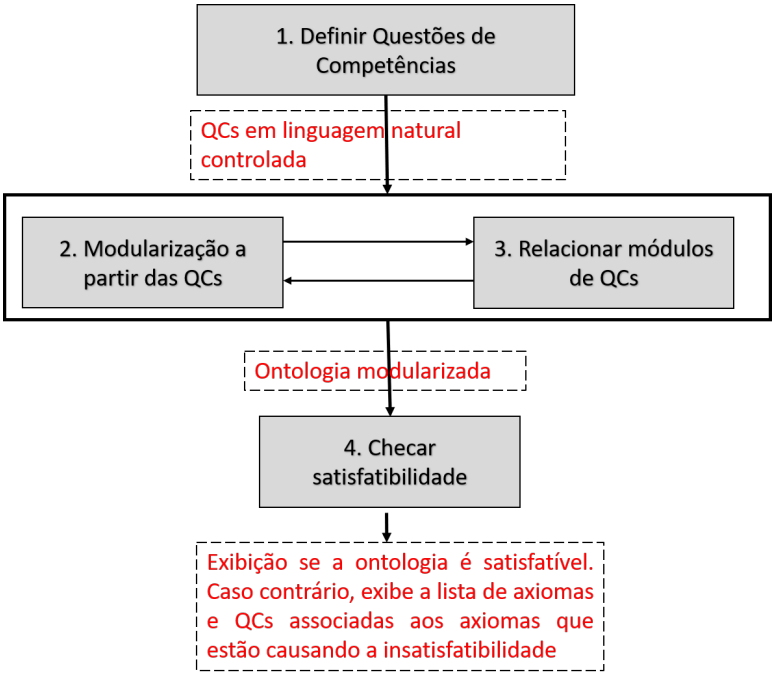


Figura 21 – Abordagem

3. É verificado também se existe alguma QC já inserida. Existindo, elas serão conectadas, e consequentemente, os módulos;

4. Por fim, cada módulo e o conjunto de módulos é checado com relação à satisfatibilidade, descobrindo, dessa forma, ainda na fase inicial, erros de especificação.

Na abordagem proposta, durante a fase de especificação, quando um desenvolvedor define uma questão de competência em linguagem natural, a QC representará um módulo, com o vocabulário e os axiomas necessários para respondê-la corretamente.

Considerar a modularização de QCs na etapa inicial provê benefícios ao desenvolvedor, como:

- Checar inconsistências entre QCs. Isso pode garantir que módulos não se tornem inconsistentes;
- Permitir um melhor entendimento das QCs, principalmente em ontologias muitos grandes;
- Permitir o reuso desses grupos de QCs associado aos respectivo conjunto de axiomas;
- Prover uma melhor manutenção da ontologia, já que as QCs estarão modularizadas e a satisfatibilidade verificada na etapa inicial do desenvolvimento.

Em nossa abordagem, partimos do pressuposto de que não há uma ontologia e que a equipe de desenvolvimento está se baseando em alguma metodologia, na qual, existe a

necessidade de definir questões de competência. No entanto, é possível após o processo de construção da ontologia através da abordagem, editá-la, adicionando novas QCs. Após a edição, todo o conjunto de QCs é checado com relação à consistência.

Como dito, uma questão de competência é um módulo que pode conter outras subQCs ou subMódulos. A Figura 6.2 exibe um exemplo de modularização de QCs e as relações entre elas. Podemos perceber que um módulo pode conter outras QCs, denominadas subQCs, além disso, elas compartilham a mesma QC, como a QCA, que é compartilhada pelos módulos QC5 e QC8. Além disso, a Figura ilustra os módulos abaixo:

- QC5 é um módulo que contêm QC1.1 e QCA;
- QC8 é um módulo que contêm QC2, QC3 e QCA;
- QC1.1 contêm os módulos QC1.2 e QC 1.1;
- QC1.1 contém o módulo QC1.1.1;
- QC2 contém o módulo QC2.1;
- QC3 contém o módulo QC3.1;
- QC3.1 contém o módulo QC3.1.1;
- QC3.1.1 contém o módulo QC3.1.1;

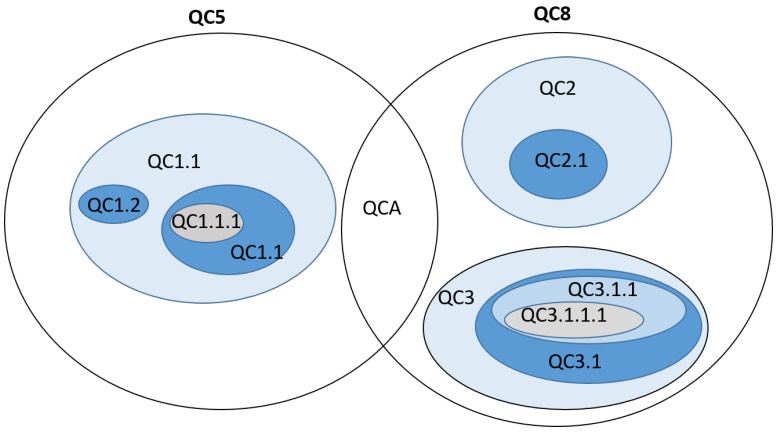


Figura 22 – Modularizaçao de CQs

Para alcançar um melhor entendimento nas próximas seções, primeiro explicaremos duas ontologias que são usadas como referência para descrever o processo e os exemplos apresentados.

6.2.1 Ontologia de Pizzas

A ontologia de Pizzas descreve o domínio de pizzas e está disponível no site do Protégé (RECTOR et al., 2004). A Figura 23 apresenta brevemente a Ontologia da Pizza. A seta roxa indica dependências de superclasse / subclasse. Assim, a classe "DomainConcept" tem "Food" e "Country" como subclasses. Por outro lado, "PizzaTopping", "Pizza" e "PizzaBase" são subclasses de "Food".

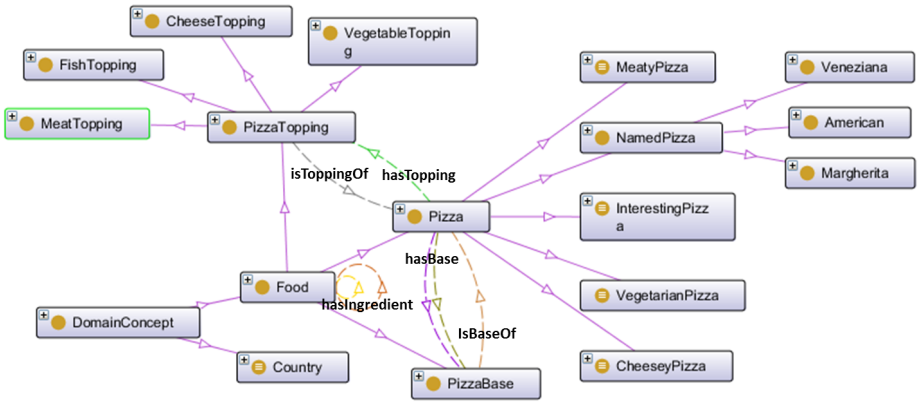


Figura 23 – Ontology Pizza

A seguir, são exibidos fragmentos da ontologia de pizza em LD:

Pizza $\sqsubseteq$ Food

PizzaTopping $\sqsubseteq$ Food

Pizza $\sqsubseteq \exists$ hasBase.PizzaBase

Country $\equiv$ { America, Italy, Germany, France, England}

CheeseyPizza $\equiv$ Pizza $\sqcap \exists$ hasTopping.CheeseTopping

MeatPizza $\equiv$ Pizza $\sqcap \exists$ hasTopping.MeatTopping

VegetarianPizza $\equiv$ Pizza $\sqcap \neg(\exists$ hasTopping.FishTopping$) \sqcap \neg(\exists$ hasTopping.MeatTopping)

InterestingPizza $\equiv$ Pizza $\sqcap \leq$ 3hasTopping

American $\sqsubseteq$ NamedPizza $\sqcap \exists$ hasCountryOfOrigin.{*America*}

6.2.2 Ontologia MadCows

A ontologia MadCows ((DAVIES; STUDER; WARREN, 2006)) descreve brevemente o domínio de animais, inclusive de vacas. A Figura 24 apresenta brevemente a Ontologia MadCows ((DAVIES; STUDER; WARREN, 2006)).

Figura 24 – Ontologia MadCow

A seguir, são exibidos fragmentos da ontologia MadCows em LD:

Grass $\sqsubseteq$ Plant
Cow $\sqsubseteq$ Vegetarian
MadCow $\sqsubseteq$ Cow
MadCow $\sqsubseteq \exists$ eat.(Brain $\sqcap \exists$ partof.Sheep)
Sheep $\sqsubseteq$ Animal
Vegetarian $\sqsubseteq \forall$ eat.($\neg$ Animal $\sqcup \neg$ ($\exists$ partof.Animal))

Na próxima seção é apresentada a metodologia utilizada.

6.3 METODOLOGIA

O primeiro passo é a criação de uma QC em linguagem natural controlada.

Verifica se, para responder à QC, são necessárias outras QCs. Por exemplo: considere a QC “Which are the toppings of Margherita pizza?”. Para obter uma resposta, precisamos responder:

- “What is a pizza?”
- “What is a Margheritapizza?”
- "What are toppings?"

Essas últimas três questões são consideradas subQCs da QC original. O módulo M consistirá da QC, sua resposta, e suas respectivas subQCs transformadas em um conjunto de axiomas.

Para cada QC definida, é verificada se ela possui subQCs, que são QCs necessárias para responder à QC inicial, como dito anteriormente. Ademais, um módulo pode possuir alguma relação com outro módulo, caso alguma subQC ou a QC inicial já esteja definida em outro módulo, dessa forma evitando redundâncias. A Figura 25 exibe uma QC modularizada, na qual para se responder a QC3, é preciso responder à QC3.1 e à QC 3.1.1, e à QC 3.1.1.1.

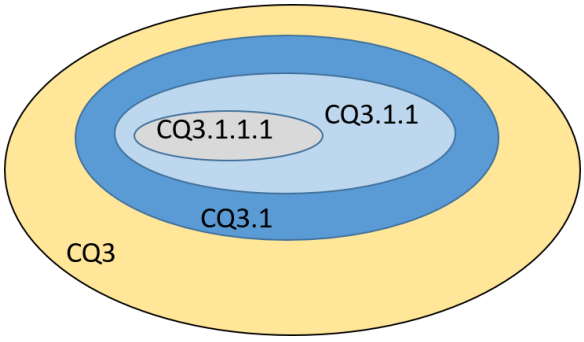


Figura 25 – Uma QC modularizada

À medida que as QCs são criadas, se já existir uma QC em outro módulo, é criada uma conexão entre esses módulos. No caso da ontologia de Pizza, existe uma QC que conecta dois módulos, como mostra a 26. Essa QC “What is topping?” é subQC das duas QCs exibidas na imagem “Which are the topping of margherita pizza?” e “Which are the toppings of ice cream?”.

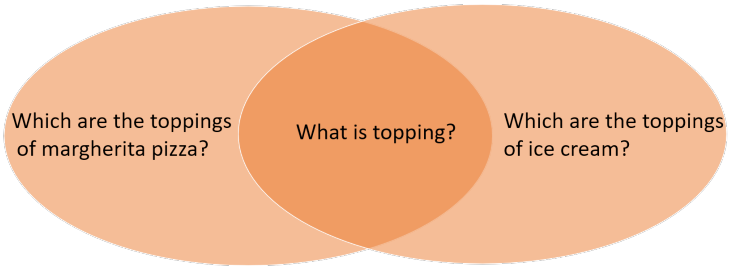


Figura 26 – QC que conecta as QCs Pizza e IceCream

A cada QC modularizada, será verificada com relação à satisfatibilidade, ou seja, descobrir se existem conjuntos de QCs insatisfatíveis e entre si, desse modo, evitando que módulos também sejam insatisfatíveis.

Na próxima seção, serão discutidos a linguagem natural controlada, seu processamento e os padrões adotados na abordagem.

6.4 PROCESSAMENTO DE LINGUAGEM NATURAL

Processamento de linguagem natural (PLN) é um área da inteligência artificial, que se concentra em analisar a linguagem natural humana para aplicar métodos computacionais, com o intuito de que a linguagem natural humana possa ser compreendida por uma máquina.

PLN é basicamente constituída nas seguintes etapas:

1. Análise morfológica
2. Análise Sintática
3. Análise Semântica
4. Análise Pragmática
5. Análise de Discurso

De forma sucinta, explicaremos as três primeiras etapas de PLN na próxima seção. Com relação à análise pragmática e de discurso, segue abaixo uma definição de cada, respectivamente:

A pragmática é um

> "ramo da linguística que analisa o uso concreto da linguagem pelos falantes da língua em seus variados contextos. A Pragmática extrapola a significação dada às palavras pela semântica e pela sintaxe, observando o contexto extra-linguístico em que estão inscritas; ou seja, ocupa-se da observação dos atos de fala e suas implicações culturais e sociais"[1].

A análise de discurso é

> "uma prática da linguística no campo da Comunicação consiste em analisar a estrutura de um texto e, a partir disso compreender as construções ideológicas presentes nele"[2].

Na próxima seção, explicaremos com mais detalhes a análise morfológica, sintática e semântica.

6.4.1 Análise Morfológica

Nessa etapa, segundo ((GORSKI; COELHO, 2010)) reconhece as palavras em termos de unidades que as compõem. São examinados e categorizados cada palavra e símbolo de uma frase. Cada palavra é classificada de acordo com seu tipo de uso ou, em linguagem natural ou categoria gramatical .

6.4.1.1 Categorias Sintáticas

As palavras em linguagem natural (LN) pertencem a uma determinada categoria sintática ou classes de palavras da língua (substantivos, adjetivos, verbos, advérbios, determinantes, numerais e conectivos), definindo o seu comportamento sintático básico.

O léxico de acordo com (INDURKHYA; DAMERAU, 2010) é definido c omo u m conjunto de categorias cognitivas (um inventário de categorias e subcategorias cognitivas; e de traços semânticos inerentes) e traços derivados que são representados nas palavras por meio da lexicalização.

Em princípio, cada gramática pode usar seu próprio conjunto de rótulos de categoria sintática, também conhecido como Tag (etiquetas), tanto para significados lexicais (léxico), como para as categorias sintagmáticas. Para a língua inglesa, por exemplo, temos um conjunto denominado *Tag Penn Treebank* ((MARCUS et al., 1994)). O *Tag Penn Treebank* é um corpus em língua inglesa. Por exemplo, na Figura 27 podemos ver, acima de cada palavra, uma marcação com etiquetas das categorias léxicas.

Para cada palavra da sentença, é encontrada a classe gramatical da palavra. Para a frase do exemplo "Vegetarian is an animal that does not eat an animal or (part of an animal)", de acordo com o *Tag Penn Treebank*, cada palavra e símbolo será classificado como:

[1] https://www.portugues.com.br/redacao/pragmatica.html
[2] https://www.infoescola.com/linguistica/analise-do-discurso/

Figura 27 – Análise léxica de uma sentença

- **Vegetarian**: NNP, substantivo próprio no singular (Proper noun, singular)
- **is**: VB, verbo, forma verbal (Verb, base form)
- **an**: DT, determinador (Determiner)
- **animal**: NN, substantivo no singular (Noun, singular)
- **that**: WDT, pronome interrogativo (Wh-determiner)
- **not**: RB, advérbio (Adverb)
- **eat**: VB, verbo, forma verbal (Verb, base form)
- **an**: DT, determinador (Determiner)
- **animal**: NN, substantivo no singular (Noun, singular)
- **or**: CC, conjunção coordenativa (Coordinating conjunction)
- **part**: NN, substantivo no singular (Noun, singular)
- **of**: IN, preposição (Preposition or subordinating conjunction)
- **an**: DT, determinador (Determiner)
- **animal**: NN, substantivo no singular (Noun, singular)

Para mais detalhes sobre *Tag Penn Treebank* ver apêndice C.

6.4.2 Análise Sintática

A análise sintática tem como objetivo estudar o papel de cada palavra dentro de uma frase ou oração e suas relações entre si. Ela se baseia em uma determinada gramática que segundo (GORSKI; COELHO, 2010) é um formalismo desenvolvido para descrever as regras sintáticas ou regularidades, de acordo com as quais, se constroem palavras e sentenças em uma língua.

O termo "sintagma" designa uma sequência de elementos linguísticos organizados de forma hierarquizada, que compõem uma unidade na sentença. Os sintagmas[3] denominam-se:

[3] https://brasilescola.uol.com.br/gramatica/tipos-sintagmas.htm

1. Sintagma Nominal: o núcleo é um substantivo;
2. Sintagma Verbal: o núcleo é um verbo;
3. Sintagma Proposicional: o núcleo é uma preposição;
4. Sintagma Adjetival: no qual o núcleo é um adjetivo;
5. Sintagma Adverbial: onde o núcleo é um advérbio.

Uma árvore sintática é uma maneira simples de visualizar uma análise sintática. Ela é uma árvore cujos "galhos" indicam a hierarquia constituída entre os sintagmas e palavras da sentença, como mostra a Figura 28.

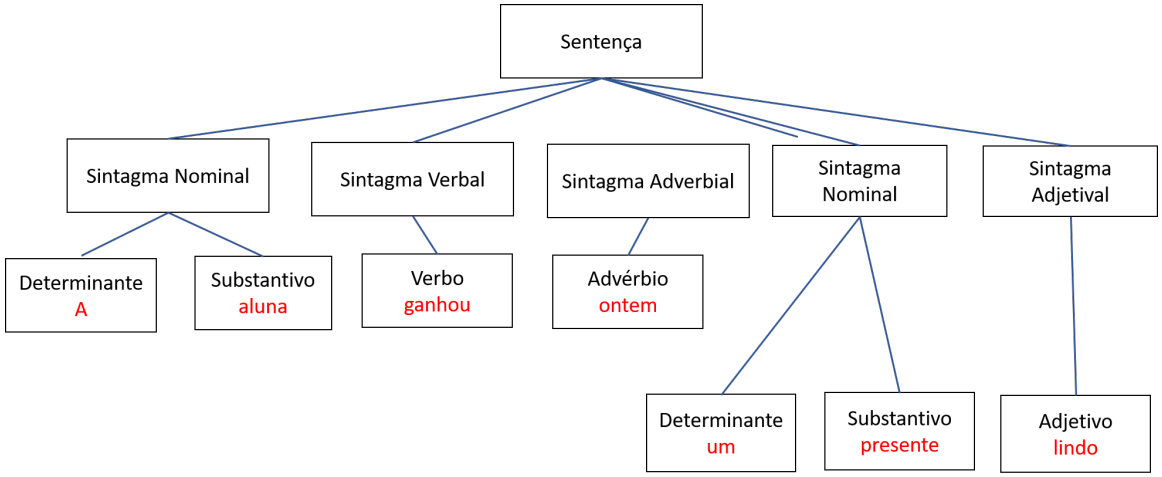


Figura 28 – Análise Sintática de uma sentença

6.4.3 Semântica

O objetivo dessa etapa é encontrar o significado da sentença. Em relação ao aspecto semântico de uma língua, podem-se destacar três propriedades [4]:

1. Sinonímia: consiste na parte da semântica que estuda as palavras sinônimas, com significados semelhantes. Por exemplo:
   - A garota renunciou ao pedido da sua família.
   - A menina recusou ao pedido da sua família.
     Percebe-se que, tanto a "garota", como a "menina" apresentam o mesmo significado. Assim como os verbos "renunciou" e "recusou".
2. Antonímia: consiste no estudo das palavras antônimas, que apresentam significados opostos. Por exemplo:

[4] https://brasilescola.uol.com.br/portugues/semantica.htm

- A garota renunciou ao pedido da sua família.
- A senhora aceitou ao pedido da sua família.

Percebe-se que tanto “garota” como “senhora” apresentam signficados diferentes. Assim como os verbos “renunciou” e “aceitou”.

3. Polissemia: estuda as palavras que possuem mais de um significado.

   - O rapaz não conseguiu sentar-se porque o banco estava quebrado.
   - O rapaz foi ao banco efetivar um depósito em sua conta.

   Percebe-se que a palavra “banco” tem significados diferentes, a partir de um contexto diferente das frases.

6.4.4 Processamento da linguagem natural controlada

Linguagem natural controlada é aquela baseada em linguagem natural, mas aplica restrições no vocabulário, gramática e/ou semântica. Tem como principal objetivo eliminar possíveis problemas do vocabulário natural, como a ambiguidade. Além disso, torna o processo de compreensão por sistemas computacionais mais viável.

Neste trabalho, não usaremos dicionário de dados para auxiliar na abordagem e na implementação dela. Em vez disso, iremos detectar padrões pré-estabelecidos para nomes de classes e de propriedades. Analisamos, por exemplo, que é mais comum utilizar substantivos e adjetivos, ou mesmo a junção dos dois para denominar uma classe. O mesmo ocorre com a utilização de verbos em propriedade.

Até o momento, trabalhamos apenas com QCs, em âmbito terminológico Tbox, pois como mostrado no capítulo, a maioria dos trabalhos foi feito apenas no âmbito assercional (ABox), utilizando ABox, o que limita o poder de checagem de ontologias utilizando QCs. Também não estamos utilizando as outras fases de Processamento de Linguagem Natural (PLN), como análise semântica, análise pragmática e de discurso, pois a análise sintática é suficiente para os objetivos do trabalho.

Na próxima seção apresentaremos a gramática utilizada na abordagem para criação de QCs.

6.4.5 Grámatica da linguagem natural controlada

Como estamos utilizando linguagem natural controlada, criamos uma gramática para escrita de QCs, ou seja, para a pergunta e a respectiva resposta. Entendemos que uma gramática consiste de um vocabulário, que está em negrito do lado direito das regras, e um conjunto de regras para a definição de uma linguagem.

QC → {*Pergunta,Resposta* }
Pergunta → <Pronome interrogativo>|<Verbo auxiliar> + <words> ***?***
Pergunta → ***What*** + <words> + <conceito> + ?
Pergunta → ***Which*** + <words> + <propriedade> + <words> + <conceito> + ***?***
Pergunta → ***Where*** + <words> + conceito + ?
Pergunta → ***Who*** + <words> + <propriedade> + <words> + <conceito> + ***?***
Pergunta → ***How many*** + <words> + <propriedade> + <words> + <conceito> + ***?***
Pergunta → <Verbo auxiliar> + <words> + conceito + ?
<Verbo auxiliar> → ***Do/Does/Did/Is/Are***
<palavras> → ***palavra***+...+**palavra**|λ
<conceito> → ***letra***|***palavra***
<propriedade> → ***palavra***
Resposta → <conceito>|<conceito>+...+<conceito>
Resposta → <conceito> + ***(is a/ is an/ is)*** + <conceito> + ***that*** + <propriedade> + ***and*** + ***only*** + ***(*** + <conceito>+ ***and*** +<conceito> ***)***
Resposta → <conceito> + ***(is a/an)*** + <conceito> + ***that*** + ***does not/do not*** + <propriedade> + ***only*** + <conceito>
Resposta → <conceito> + ***(is a/an)*** + <conceito> + ***that*** + <propriedade> + <conceito>
Resposta → <conceito> + ***(is a/an)*** + <conceito> + ***that*** + ***does not/ do not*** + <propriedade> + ***only*** + ***(*** + <conceito> + ***and/or*** + <propriedade> + <conceito> + ***)***
Resposta → <conceito> + ***(is equivalent to/a/an)*** + <conceito> + ***that*** + ***is*** + **not** + <conceito>
Resposta → <conceito> + ***(is equivalent to /a/an)*** + <conceito> + ***that*** + ***not*** + <propriedade> + <conceito> + ***and*** + <conceito>
Resposta → <conceito> + ***(is equivalent to /a/an)*** + <conceito> + ***or*** + <conceito> + ***or*** + <conceito> + ***or*** + <conceito>
Resposta → <conceito> + ***(is equivalent to /a/an)*** + <conceito> + ***that*** + <propriedade> + ***only*** + <conceito>
Resposta → <conceito> + ***(is equivalent to /a/an)*** + <conceito> + ***that*** + <propriedade> + <conceito>
Resposta → <conceito> + ***(is equivalent to /a/an)*** + <conceito> + ***that*** + <propriedade> + <conceito> + ***and/or*** + ***(*** + <propriedade> + <conceito> + ***)***
Resposta → <conceito> + ***is different of*** + <conceito>
Resposta → ***domain/range*** + ***of property*** + <property> + ***is*** + <conceito>
Resposta → Resposta + ... + Resposta

Na próxima subseção apresentaremos os padrões adotados na abordagem para obten-

ção dos conceitos e relações das sentenças.

6.4.5.1 Padrões adotados para encontrar conceitos e relações

Para melhor entendimento, considere a QC, na qual a pergunta consiste de "What is a vegetarian animal?" e a resposta "vegetarian is an animal that does not eat (animal or (part of animal))".

Quando o desenvolvedor insere uma QC na ferramenta, ela gera subQCs de acordo com a detecção de substantivos e adjetivos no QC, as quais são conceitos (classes) em potenciais.

1. Para cada conceito encontrado, é gerada uma subQC "What is a C?", onde C é:
   Substantivo+Substantivo, Substantivo+Adjetivo, Substantivo,
   Adjetivo+Substantivo
   Conceitos detectadas: Vegetarian e Animal

2. Similarmente, procuramos por propriedades, que podem ser:

   Verbo, Verbo+Preposição, Substantivo+Preposição
   Propriedades detectadas: eat, partOf

Em outro exemplo, para a QC "Which are the pizzas that have mozzarella topping but does not have meat topping?", seguindo esse padrão, conseguimos identificar que *pizza*, *MozzarellaTopping e MeatTopping* são classes; e *have* é uma propriedade.

Existem algumas regras para escrever uma QC, uma vez que estamos lidando com linguagem natural controlada. Alguns exemplos são mostradas na Tabela 3.

⇒ Para tornar os padrões mais semelhantes à linguagem natural, o quantificador existencial ∃ não é necessário definir nas descrições de axiomas, já que não é comum dizer, por exemplo, "Este sorvete tem algum morango e algum chocolate". No entanto, é necessário colocar o existencial universal ∀.

Tabela 3 – Tabela de exemplos dos padrões de escrita de axiomas

Exemplos
X is a pizza that has mozzarella and only (Tomato and Garlic)?
X is equivalent a Pizza that is not Vegetarian
X is equivalent a Pizza that not has Meat and Fish.
X is equivalent to Picante or Media or NonPicante
X is equivalent to pizza that has only baseline
Does AmericanPizza have MozzarellaTopping?
X is a Food that has Fruit
X is diffrent of Y
X is a pizza that not has only Tomato
Brain is an organ that part of animal
MadCow is equivalent to Cow that eats Brain and (part Of Sheep)
Vegetarian is an animal that does not eat only (animal or (part Of animal)
domain of eats is animal

6.5 PROCESSO DE MODULARIZAÇÃO

Em nossa abordagem, cada QC é transformada, junto com sua resposta, num módulo M que consiste em um conjunto de axiomas terminológicos. Consideramos QCs expressas como sentenças interrogativas. A diferença entre os tipos de QC está relacionada ao tipo de resposta obtida pela QC quando consultamos a ontologia. Nós identificamos três tipos principais:

1. **QCs que trabalham sobre classes e suas relações**. Aqui, a resposta consiste em classes ou instâncias. Por exemplo, para a QC “What is the base of a Real Italian Pizza?”, a classe “ThinAndCrispyBase” será retornada de acordo com a ontologia de Pizzas;

2. **Problemas de decisão expressos como QCs**. Nesse tipo, a resposta permitida para a pergunta só pode ser verdadeira ou falsa. Por exemplo, para a QC “Is pizza a food?”, “yes” será a resposta, de acordo com a ontologia de Pizza;

3. **QCs com resposta equivalente a um conjunto de axiomas**. Nesse tipo, por exemplo, a QC é “What is a MeatyPizza?” terá como resposta um conjunto de axiomas que definem a “MeatyPizza”, como a descrição abaixo:

$\text{MeatyPizza} \equiv \text{Pizza} \sqcap \exists\ \text{hasTopping.MeatyTopping}$
$\text{MeatyTopping} \sqsubseteq \text{PizzaTopping}$

Que significa que toda pizza de carne (*MeatyPizza*) equivale a uma Pizza e existe cobertura de carne. Além disso, o conceito *MeatTopping* é um *PizzaTopping*.

6.5.1 Etapas detalhadas do processo

A Figura 29 exibe as etapas do processo de modularização de QCs por meio de composição. Primeiro, o usuário cria um projeto, que irá abrigar a ontologia a ser gerada e dados sobre a modularização. Depois, ele irá inserir as QCs, uma por uma, e cada QC se transformará em um módulo. Deve-se, então, verificar se existem insatisfatibilidade dentro de um módulo ou entre módulos.

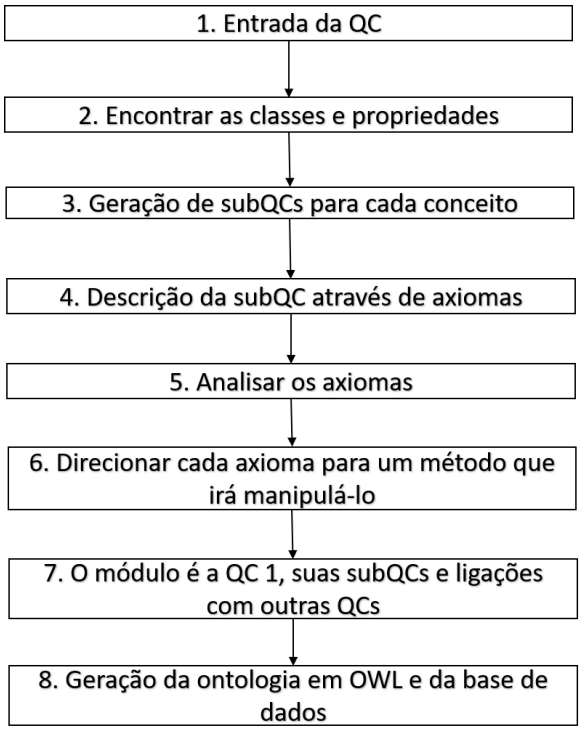


Figura 29 – Passos da metodologia

Para uma melhor compreensão do processo, considere as seguintes QCs:

1. **QC1: "Which are the toppings of a Lareine pizza?"**
2. **QC2: "Which are the toppings of vegetarian pizza?"**

E, segundo a ontologia de pizza, as resposta são, respectivamente:

1. *"LaReine é a Pizza and has topping ham, mozzarella, tomato and olive"* e,

2. *"Vegetarian equivalent a Pizza and not has topping fish and meat".*

O processo segue, basicamente, os passos abaixo:

1. Para cada conceito encontrado na QC, é gerada uma subQC "What is a/an C?", considerando as QC1 e QC2:
   Para **QC1**:

   - What is a pizza?
   - What is a topping?
   - What is a lareine?

   Para **QC2**:

   - What is a topping?
   - What is a vegetarian?
   - What is a pizza?

   ⇒ *O usuário possui a opção de não responder as subQCs geradas, caso ele julgue que não faz parte do escopo da ontologia, como no caso da última pergunta.*

   Além disso, "C" pode ser:

   - Substantivo+Substantivo, Substantivo+Adjetivo, Substantivo, Adjetivo+Substantivo
     Para **QC1**: *VegetarianPizza* casa com o padrão Adjetivo+Substantivo, que consiste numa classe.
     Assim como, *Pizza*, *fish* e *meat*.
     Para **QC2**: *LaReine*, *ham*, *mozzarella*, *tomato e Olive* são classes.

2. Similarmente, procuramos por propriedades, que podem ser:

   - Verbo, Verbo+Preposição, Substantivo+Preposição
     "has" equivale a um verbo que é a propriedade encontrada, tanto na QC1, como na QC2.

3. Neste passo, o desenvolvedor, para cada subQC, pode inserir axiomas em linguagem natural controlada relacionados à questão de competência. Nesse caso, a resposta para QC é um conjunto de axiomas.

   ⇒ Os axiomas são analisados, tratados e direcionados para um metódo que irá tratá-lo.

4. Se a ferramenta detectar que, para as respostas da QC atual, existe outra QC ou subQC em outro módulo existente, os módulos são conectados através dessas QCs existentes em ambos os módulos, evitando assim, redundância.

   ⇒ No exemplo, temos duas QCs iguais que fazem parte da QC1 e QC2: "What is a topping?" e "What is a pizza?". Sendo assim, essas QCs serão conectadas, assim como os módulos nos quais elas estão contidas.

5. Depois de o usuário inserir outras QCs, uma ontologia é criada, baseada na modularização das QCs.

6. Além disso, é criado um banco de dados com informações sobre a modularização.

Para cada questão e sua resposta, existe uma gramática com regras de formação para escrita da QC, como mostrado anteriormente. Para cada QC, será gerada uma ontologia e os dados da modularização.

6.6 CHECAGEM DE SATISFATIBILIDADE

Abordagem também considera a importância de se verificar se existem insatisfatibilidades entre módulos. A contribuição importante aqui é detectar insatisfatibilidades no início do ciclo de desenvolvimento, dessa forma, economizando tempo e trabalho futuro. É importante frisar que as insatisfatibilidades podem ocorrer em QCs mal definidas pelo desenvolvedor.

As insatisfatibilidades são detectadas entre QCs, consequentemente entre módulos, ou dentro do mesmo módulo. O usuário pode verificar a satisfatibilidade dos módulos. Nós usamos um raciocinador e, se for encontrada alguma insatisfatibilidade, a abordagem proposta faz uma correlação com as QCs que geraram essa insatisfatibilidade. Portanto, detecta as QCs contraditórias.

Por exemplo, considere duas QCs:

QC1 = "**Which are the toppings of Margherita pizza?**", a resposta "**Tomato**" e a propriedade "**has**", de acordo com os axiomas inseridos pelo usuário. A referida QC gerará as seguintes subQCs:

- What is pizza?

  A resposta seria: Pizza is a Food
  $Pizza \sqsubseteq Food$

- What is a MargueritaPizza?

A resposta seria: MargheritaPizza is a Pizza that has topping only Tomato
MargheritaPizza $\sqsubseteq$ Pizza $\sqcap \forall$ has.Tomato
$\top \sqsubseteq \forall .has^{-}$.MargheritaPizza
$\Rightarrow$ Para *domains* e *ranges* é gerado uma QC, neste caso, a QC "What is the domain of Pizza?" que equivale a MargheritaPizza.

- What are toppings?

A resposta seria: Pizzatopping is a Food and not a Pizza
PizzaTopping $\sqsubseteq$ Food $\sqcap \neg$ Pizza

Depois, para a **QC2 "Which are the toppings of an icecream?"**, a resposta **"Lemon and Mint"** e a propriedade **"has"**. As as subQCs geradas são as seguintes:

- What is icecream?

A resposta seria: Icecream is a Food that has topping lemon and mint and disjoint of Pizza
Icecream $\sqsubseteq$ Food $\sqcap \exists$ has.Lemon $\sqcap \exists$ has.Mint
Icecream $\sqsubseteq \neg$ MargheritaPizza

- What are toppings?

A resposta seria: Considerando que, a QC1 "Which are the toppings of Margherita pizza?" possui "What are toppings?", a QC1 será reusada em QC2, criando uma conexão entre elas.

Existe uma insatifiabilidade entre a QC1 e a QC2, porque o domínio da propriedade **"has"** é **"Pizza"**, mas **"Icecream"** usa essa propriedade também, sendo que a classe **"Pizza"** e **"Icecream"** são disjuntas.

Na abordagem proposta, essa insatisfatibilidade será detectada como também as QCs relacionadas com os axiomas que causaram essa insatisfatibilidade, que consequentemente, tornará a ontologia inconsistente.

Axiomas que causaram insatisfatibilidade:
Is Icecream satisfiable? : false

Icecream $\sqsubseteq \neg$ Pizza
Icecream $\sqsubseteq$ Pizza $\sqcap \exists$ has.Lemon $\sqcap \exists$ has.Mint
$\top \sqsubseteq \forall .has^{-}$.MargheritaPizza

Questões de Competência que representam os axiomas que geraram a insatisfatibilidade:

1. Are Icecream and MargheritaPizza disjoint? Answer= yes

2. What is icecream? / Answer= Icecream is a Food that has lemon and mint

3. Which is the domain of property has? Answer= MargheritaPizza

6.6.1 Definições formais

Definição 25 ***(Questões de Competência)****. Uma questão de competência é um par* $\langle Q^{NL}, \delta^{NL} \rangle$*, considerando que NL significa* Natural Language*, tal que* Q^{NL} *é uma consulta expressa em uma linguagem e* δ^{NL} *é uma resposta para esta consulta expressa como uma variável de substituição.*

Definição 26 ***(Questão de competência satisfeita)****. Uma questão de competência* $\langle Q, \sigma \rangle$ *é satisfeita por uma ontologia* O *se* $O \models \sigma(Q)$*.*

Definição 27 ***(Módulo insatisfatível)****. Uma ontologia ou módulo é insatisfatível se ela contém em pelo menos dois módulos* M_1 *e* M_2 *representando QC1 and QC2 com seu respectivo conjunto de axiomas* $\sum_1$ *e* $\sum_2$ *tal que* $\sum_1 \cup \sum_2 \models \bot$

Definição 28 ***(QCs contraditórias)****. Duas QCs que juntas causam insatisfatibilidade em uma ontologia ou módulo no qual elas são partes, são ditas QCs contraditórias.*

Para validação da abordagem foi construída uma ferramenta que é semi-automática, descrita na próxima seção.

7 VALIDAÇÃO COM A CQSETCONSISTENCYCHECKER E EXPERIMENTOS

Neste capítulo é apresentada a ferramenta para validação da proposta denominada CQSetConsistencyChecker, e será descrito como foram realizados os experimentos dessa validação.

7.1 CQSETCONSISTENCYCHECKER

Para testar nossa abordagem, criamos uma ferramenta chamada CQSetConsistency Cheker. A Figura 30 exibe a arquitetura da CQSetConsistencyChecker. A ferramenta utiliza APIs, como a API OWL (HORRIDGE; BECHHOFER, 2011), que é necessária em todos os módulos da ferramenta para extrair informações como axiomas e classes da ontologia. A Standford NLP API ((MANNING et al., 2014)) é empregada para tratamento do processamento da linguagem natural.

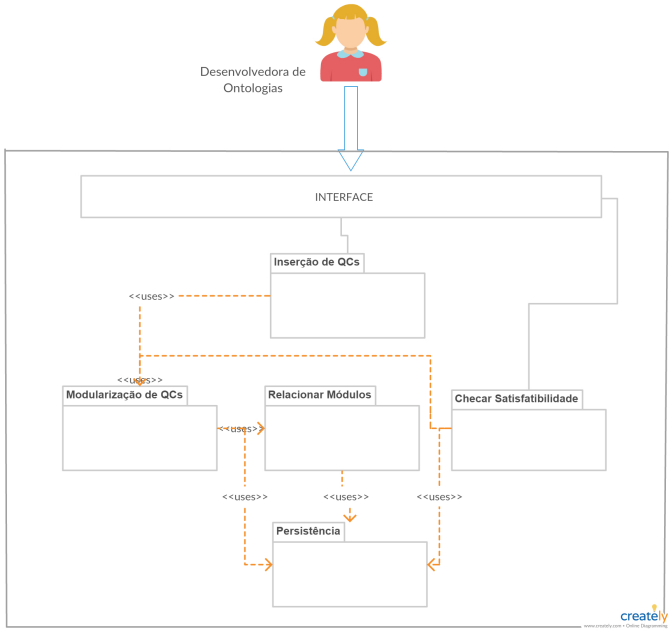


Figura 30 – Arquitetura do trabalho proposto

Cada pacote da arquitetura é responsável por implementar uma funcionalidade da abordagem proposta. Abaixo a descrição de cada pacote:

- Inserção de QCs: o módulo trata as QCs em linguagem natural controlada, conforme a gramática (seção 6.4.4). Além disso, verifica a QC de entrada e gera subQCs, se for o caso;
- Modularização de QCs: as QCs são modularizadas. Cada módulo corresponde à QC de entrada e as suas subQCs;

- Relacionar Módulos: se uma mesma QC fizer parte de dois ou mais módulos, esses módulos terão uma ligação entre eles;
- Checar Satisfatibilidade: checa se a ontologia modularizada apresenta insatisfatibilidades entre QCs, consequentemente, entre módulos;
- Persistência: módulo responsável pela manipulação da ontologia em OWL, bem como da base de dados gerada com as informações da modularização.

Neste trabalho, usamos apenas QCs em nível terminológico (TBox). Da mesma forma, na implementação só são aceitas QCs em linguagem natural controlada baseadas em $\mathcal{DL}$ $\mathcal{ALCN}$. Como trabalhamos com linguagem natural controlada, estamos lidando com alguns padrões de QCs mostrados no capítulo 6.

Na seção seguinte, iremos abordar o processo de modularização na ferramenta.

7.1.1 Modularização de QCs

Trata-se de um módulo da ferramenta responsável por modularizar as QCs e gerar a ontologia correspondente após o processo. Como foi dito antes, existem três abordagens de modularização: composição, extração e decomposição.

Como estamos propondo uma abordagem inovadora de modularização fazendo uso de QCs, para o usuário poder decompor e extrair módulos de uma ontologia, primeiro precisa utilizar uma ontologia existente que foi gerada utilizando a abordagem.

Nas próximas seções serão apresentadas as operações de modularização implementadas na ferramenta.

7.1.1.1 Composição de CQs

O desenvolvedor insere uma QC na ferramenta (ver Figura 31), composta pela pergunta e sua resposta. A pergunta gera subQCs de acordo com a detecção de substantivos e adjetivos, que são classes em potencial, usando a Stanford NLP API ((MANNING et al., 2014)).Além disso, detecta possíveis propriedades, considerando as limitações do processamento de linguagem natural atual.

O algoritmo para detecção dos conceitos e propriedades está na Figura 32.

1. O algoritmo recebe uma QC em linguagem natural controlada;
2. Divide a QC em tokens e a armazena numa lista de tokens;
3. Para cada token, verifica se é um verbo ou verbo+preposição ou substantivo+verbo. Se a verificação for positiva, o token é adicionado à lista de tokens de propriedades;
4. Para cada token, verifica se é um substantivo ou substantivo+substantivo ou substantivo+adjetivo. Se a verificação for positiva, o token é adicionado à lista de tokens de conceitos.

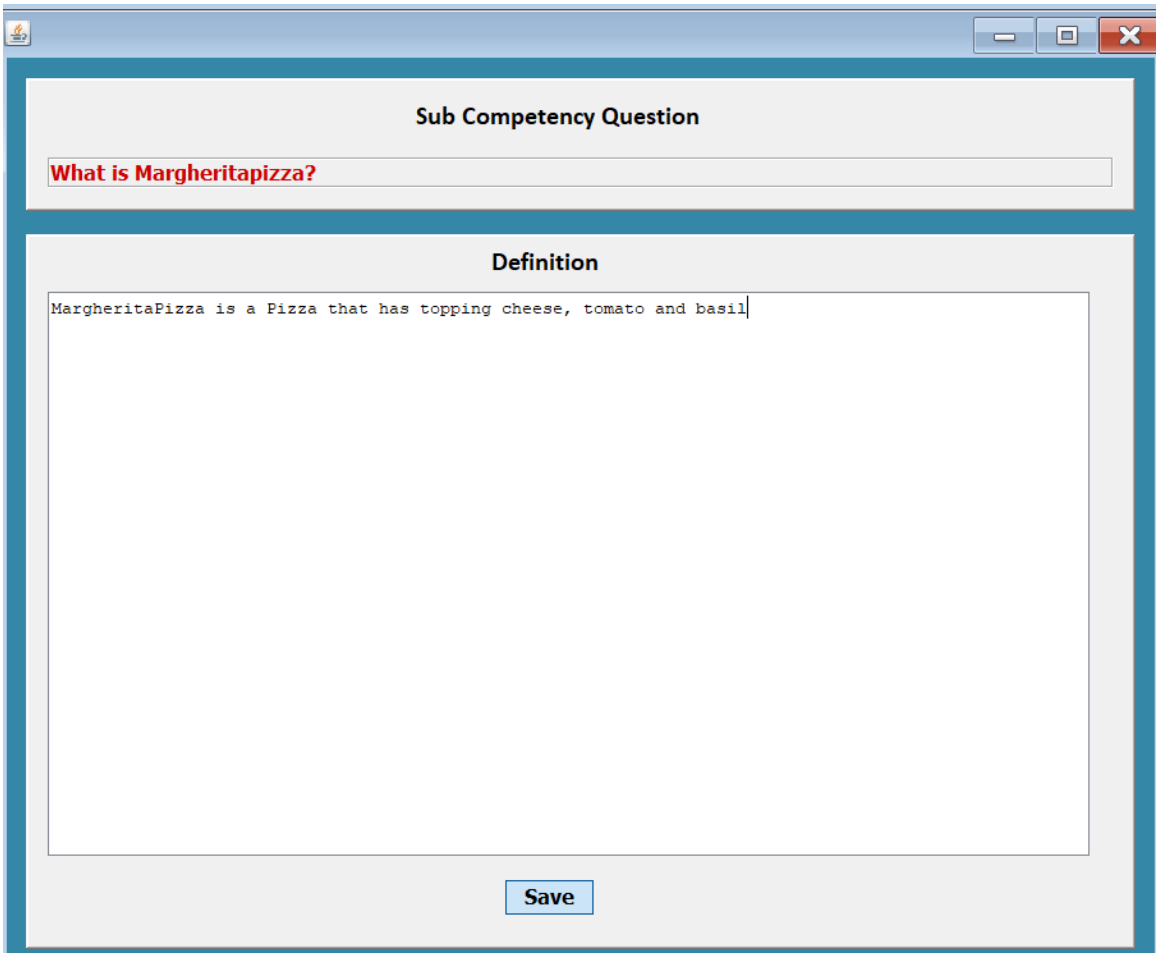


Figura 31 – Tela de inserção da CQ

Após a inserção da QC, são geradas subQCs conforme os seguintes passos:

1. Para cada conceito encontrado, é gerada uma subQC “O que é a/an C?”, onde C é um conceito;

2. São criadas QCs para o domínio e a imagem da propriedade inserida;

3. Para cada subQC, uma tela é gerada, na qual o usuário pode inserir axiomas em linguagem natural controlada relacionada à questão de competência (ver Figura 31). No caso, a resposta para a subQC é um conjunto de axiomas;

4. Para cada QC, existe um padrão, e essa QC é direcionada para um método que irá tratá-la (ver Figura 33).

5. Da mesma forma, para a descrição das respostas existem padrões, e para cada um deles, um método para tratá-los (ver Figura 33).

6. Se a ferramenta detectou que, para responder à QC atual, outra QC já existe em outro módulo, os módulos serão vinculados. Para isso, a ferramenta faz uma busca na base de QCs, verificando se a QC a ser inserida, já existe;

FindEntities: Algorithm to find roles and concepts

Inputs: CQ_{NL} (Competency questions in controlled natural language),

Output: $RoleTokenList := \{role | role\ extracted\ from\ CQ_{NL}$
ConceptTokenList := {concept| concept extracted from CQ_{NL}}

```
1. TokenSet := {token | token extracted from CQ_NL}
2. RoleTokenList :={}
3. ConceptTokenList :={}
4. For i=1 to size(TokenSet)
     a. If token[i]=verb or verb+prep or noun+verb then
          i. RoleTokenList := token[i]
        Endif
     b. Endfor
5. For i=1 to size(TokenSet)
     a. If token[i]=noun or noun+noun or noun+adj then
          i. ConceptTokenList := token[i]
        Endif
     b. Endfor
6. Endfor
```

Figura 32 – Algoritmo para encontrar conceitos e papéis

7. Depois que o desenvolvedor insere todas as QCs, uma ontologia em OWL é gerada automaticamente;

8. Além disso, é criado um banco de dados com informações sobre o processo de modularização. Usamos o SQLite ((ALLEN; OWENS, 2010)), pois não há necessidade de o usuário instalar.

O algoritmo 33 recebe uma QC em linguagem natural controlada e retorna um conjunto de axiomas. Considerando uma lista de axiomas em linguagem natural controlada, para cada axioma na lista, ele verifica qual método no programa foi criado para tratar esse axioma. O método irá analisar o axioma e transformá-lo em OWL.

É importante frisar que o desenvolvedor pode mudar os nomes das classes na descrição da subQC no campo “axiomas”, como mostra a Figura 31. Na pergunta “What is an MargheritaPizza”, o desenvolvedor poderá descrever o axioma utilizando apenas “Margherita”. O que for inserido nesse campo de axiomas será, de fato, colocado na ontologia.

Algorithm to deal with axioms

Inputs: CQ_{NL} (Competency questions in controlled natural language),

Output: $set < OWL\ axioms >$

```
1. Set:=∅;
2. CNLAxiomList := {CNLAxiom| CNLAxiom extracted from CQAxioms_NL}
3. RoleTokenList := FindEntities. RoleTokenList
4. ConceptTokenList := FindEntities. ConceptTokenList
5. PatternList := {pattern| pattern is a method that recognizes a linguistic pattern}
6. For i=1 to size(CNLAxiomList)
      a. For j=1 to size(PatternList)
            i. If match ( CNLAxiom [i] , j) then
                  1. OWLResultSet:=Transform2OWL(CNL_Axiom [i], pattern[j])
                  2. Set:=Set U OWLResultSet;
               Endif
         Endfor
   Return(Set);
7. Endfor
```

Figura 33 – Algoritmo para tratar axiomas

7.1.1.2 Extração

A extração consiste em retirar um fragmento de uma ontologia. Na ferramenta, o desenvolvedor acessa a base de dados para selecionar quais QCs ele deseja extrair (ver Figura 34).

A partir disso, a ferramenta constrói um novo conjunto de módulos acrescido da ontologia correspondente, com as QCs selecionadas, porém considerando suas subQCs e relações com outros módulos. Se a QC1 tiver alguma relação com a QC2 de outro módulo, a ferramenta irá adicionar, no novo módulo extraído, a QC2. A Figura 34 exibe a tela de extração com QCs da ontologia de Pizza ((RECTOR et al., 2004)).

Por exemplo, se QC1, que é "Which are the toppings of a margherita pizza?", depende da QC2 que é "What is a topping?", e essa QC2 pertence a outro módulo, digamos "Which are the toppings of a meaty pizza?"; e se o usuário quiser importar "Which are the toppings of margherita pizza?", a ferramenta irá gerar um novo conjunto de módulos e sua respectiva ontologia, com QC1, as subQCs de QC1 e QC2.

7.1.1.3 Decomposição

Na decomposição, o usuário utiliza uma ontologia gerada pela abordagem proposta. Sendo assim, a ferramenta decompõe a ontologia em módulos que foram previamente construídos, considerando apenas a definição de módulo abordada neste trabalho.

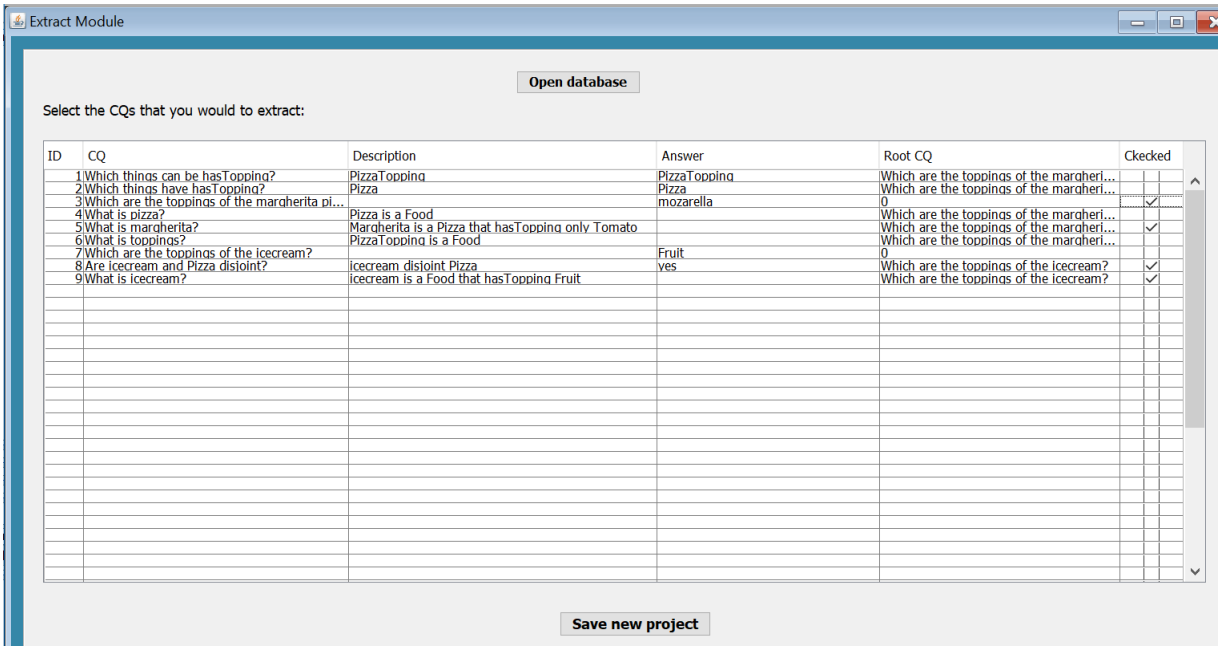


Figura 34 – Tela de extração de módulo

Para exemplificar a decomposição, considere as QCs que estão dentro de um projeto, que o desenvolvedor escolheu para decompor:

1. Which are the toppings of a margherita pizza? Tem como subQCs:
    a) What is a pizza?
    b) What is a margheritapizza?
    c) What is toppings?
2. Does an American pizza have mozzarella topping?
    a) What is a American?
    b) What is a mozzarela?
    c) What is a Pizza?
3. Which are the toppings of ice cream?
    a) What is a cream?
    b) What is a icecream?

"What is a Pizza" é compartilhada pelas QC1 e pela QC2, já que ambas apresentam esta mesma QC.

Na decomposição, serão gerados três módulos, correspondentes às três QCs:

1° módulo:

1. Which are the toppings of margherita pizza? Tem como subQCs:

a) What is a pizza?

b) What is a margheritapizza?

c) What is topping?

2° módulo:

1. Does American pizza have mozzarella topping?

   a) What is a American?

   b) What is a mozzarela?

   c) What is a Pizza?

3º módulo:

1. Which are the toppings of ice cream?

   a) What is a cream?

   b) What is a icecream?

Além disso, cada módulo é independente e está associado a um arquivo OWL diferente. Na Figura 35 é exibida a primeira QC e o fragmento da sua descrição em OWL.

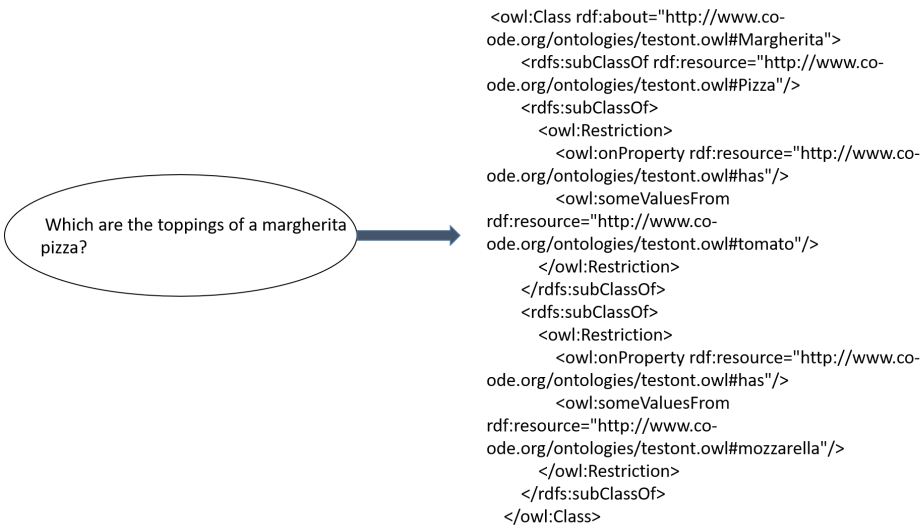


Figura 35 – QC e descrição em OWL

7.1.2 Grafo da modularização

Para a geração do grafo foi utilizada a API GraphStream, escolhida porque gera grafos interativos, nos quais o usuário pode alterar a posição do grafo, aumentar ou diminuir, alterar a posição dos nós etc.

Como exemplo, é gerado o grafo da ontologia de Pizza, ilustrado na Figura 36.

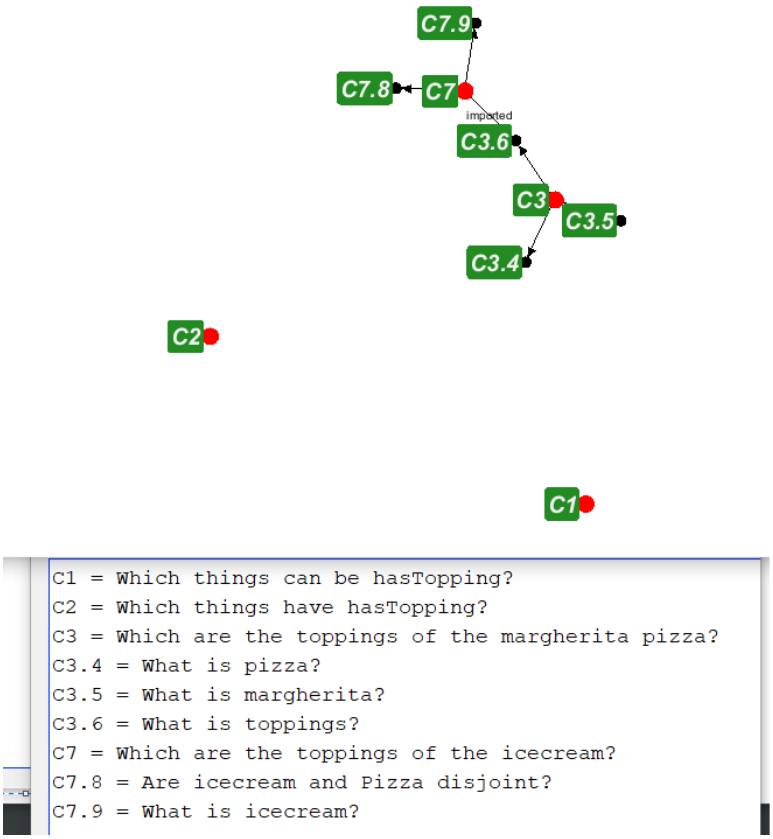


Figura 36 – Grafo da Modularização

Além de mostrar os módulos e suas relações, o grafo também mostra, na cor vermelha, módulos que estão causando insatisfatibilidade.

7.1.3 Persistência e exportação

A ferramenta também apresenta um módulo de persistência, responsável por:

- Manipular ontologias OWL;
- Manipular banco de dados;
- Geração das informações das QCs em arquivo um arquivo do tipo csv.

Na próxima seção descreveremos como os experimentos da ferramenta foram realizados.

7.2 DESCRIÇÃO DOS EXPERIMENTOS

7.2.1 Objetivo

Avaliar a abordagem proposta neste trabalho, utilizando a ferramenta implementada com desenvolvedores externos de ontologias, além de descobrir limitações para melhorar a abordagem e a ferramenta implementada.

7.2.2 Métodos utilizados

Serão utilizados os métodos de quasi-experimento e um questionário qualitativo para validar a maioria da proposta e responder às perguntas de pesquisa especificadas na seção 1.3. Segundo (CAMPBELL; STANLEY, 1963) "Os métodos quasi-experimentais constituem uma classe de estudos de natureza empírica a que falta duas das características usuais na experimentação: um controle completo e a aleatoriedade na seleção de grupos".

7.2.3 População que participará dos experimentos

Além dos próprios desenvolvedores do trabalho, os sujeitos que participaram dos experimentos serão desenvolvedores de ontologias em OWL, escolhidos de maneira não aleatória, e fazem parte de dois grupos distintos:

- Alunos da disciplina Ontologias e Web Semântica do Centro de Informática;
- Desenvolvedores de outros projetos de pesquisa.

7.2.4 Características do artefato

Serão utilizadas ontologias em OWL, preferencialmente de pequeno porte, e consequentemente um conjunto de QCs que as satisfaça.

7.2.5 Ameaças à validade do estudo

As ameaças a validade do experimento, compreendidas pelo autor desse trabalho, são:

- Presença do autor do trabalho durante o experimento;
- Diferença no perfil dos participantes com relação ao grau de conhecimento sobre engenharia de ontologias;
- Quantidade pequena de participantes;
- Instrumento de medida inadequado.

7.3 PLANEJAMENTO

7.3.1 Fase 1: Teste com os desenvolvedores do projeto

Os desenvolvedores da ferramenta irão introduzir um conjunto de QCs que serão modularizadas à medida em que forem inseridas, além da verificação de satisfatibilidade da ontologia através das QCs, com a detecção de insatisfatibilidades entre as QCs, caso haja.

Como validação preliminar, realizamos testes com três ontologias: Pizza, Viagem e Vinho, disponíveis no site do Protégé. Esses resultados preliminares nos ajudaram a melhorar a ferramenta e descobrir novos padrões controlados por linguagem natural.

Para tanto, construímos QCs para essas três ontologias através da ferramenta. Comparamos se a ontologia criada correspondia às ontologias mencionadas, como também se possuía as mesmas insatisfatibilidades.

As QCs estão relacionadas a vários constructos de $\mathcal{DL}$ $\mathcal{ALCN}$ como hierarquias de classes (A $\sqsubseteq$ B), classes disjuntas (A $\sqsubseteq \neg$ B), intersecção (A $\sqcap$ B) e união de classes (A $\sqcup$ B); classes equivalentes (A $\equiv$ B), restrições existenciais e universais. Também testamos com a conhecida ontologia [1]Mad Cows (Figura 37).

Para os conceitos Vegetarian, Cow e MadCow, temos as seguintes descrições inseridas na ferramenta:

1. Vegetariano é equivalente a um animal que (come apenas (não animal) e que come apenas (não (parte de animal)))
2. A vaca é vegetariana
3. MadCow é equivalente à vaca que come cérebros de ovelhas
4. Cérebro é uma parte de animal

Figura 37 – MadCow Ontology

No exemplo da vaca louca (MadCow), as vacas são especificadas como vegetarianas, ou seja, não comem animais, nem parte de animais. No entanto, as vacas loucas são especificadas como as que comem cérebros de ovelhas. O problema resulta da contradição entre a especificação de Cow e MadCow, que gera uma insatisfatibilidade (ver Figura 38).

7.3.2 Fase 2: Teste com desenvolvedores do trabalho de outros projetos

Foram dadas a cada participante, com o perfil descrito na seção 7.2.3, QCs para construir três ontologias: Pizza, MadCow e BreastCancer. O teste cumpriu as seguintes etapas:

1. Criar, a partir das QCs, as ontologias definidas acima;
2. Interagir com a ferramenta, inserindo as QCs que irá modularizá-las, utilizando as subQCs;
3. A ferramenta irá modularizar as QCs uma a uma, gerando módulos;

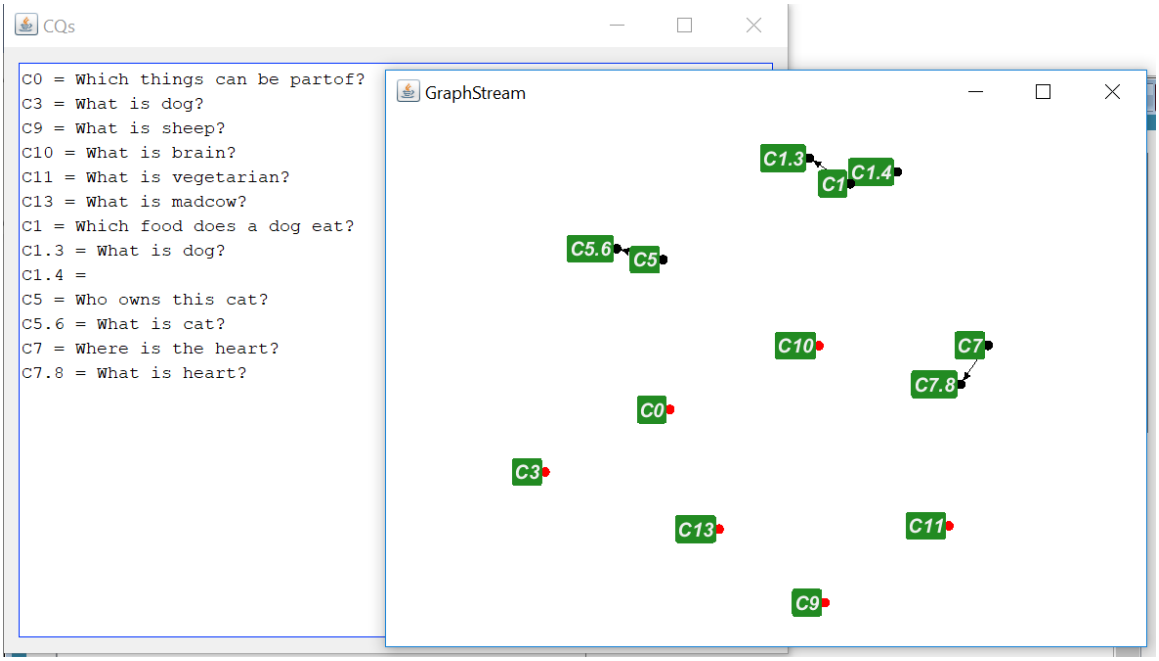


Figura 38 – Gráfico dos módulos e checagem de satisfiatibilidade

4. A ferramenta irá detectar se existem QCs repetidas. Caso existam, criará uma conexão entre elas;
5. O participante irá verificar se há insatisfatibilidade na modularização criada;
6. Visualizar o grafo com os módulos;
7. Fazer uma decomposição;
8. Fazer uma extração.

Os dois primeiros conjuntos de QCs para criar a Pizza e MadCow ontology, respectivamente, apresentam insatisfatibilidade, detectada pela ferramenta. Assim, o usuário, no inicío do desenvolvimento, saberá quais requisitos (QCs) causam inconsistências na ontologia a ser desenvolvida.

Após a interação da ferramenta, os participantes responderam a um questionário de perfil qualitativo, descrito abaixo.

7.3.3 Questionário qualitativo

Os participantes (ver seção 7.2.3) responderam a um questionário preparado pelos autores para auxiliar a busca por respostas para o problema de pesquisa. O questionário foi criado com base no trabalho de (MANZATO; SANTOS, 2012) que descreve como criar um questionário. As questões estão numa escala de 1 a 10 e são as seguintes:

1. Do you think that finding inconsistencies among the requirements in the beginning of the process is helpful for ontology engineers?

2. The approach helps a ontology developer to spare time and work?

3. Does the approach help a ontology developer to build a modular and consistent ontology?

4. Is controlled natural language an easy/suitable entry for creating axioms?

Além dessas questões, é pedido para o participante escrever sugestões para melhoria da abordagem, caso ele se sinta confortável.

Esse questionário foi aplicado a uma turma de Web Semântica e Ontologias em 2018, que era composta por 5 alunos, mais 3 pesquisadores que trabalham com ontologias. Utilizamos o formulário do Google. As Figuras 39, 40, 41, 42 e 43 7.14 exibem o resumo da avaliação pelos desenvolvedores que participaram dos experimentos. Ao final, 8 pessoas participaram do questionário.

O resultado para a pergunta "Encontrar inconsistências entre requisitos da ontologia no início do processo é útil para desenvolvedores de ontologias" é ilustrado na Figura 39, e a resposta para todos os participantes foi afirmativa.

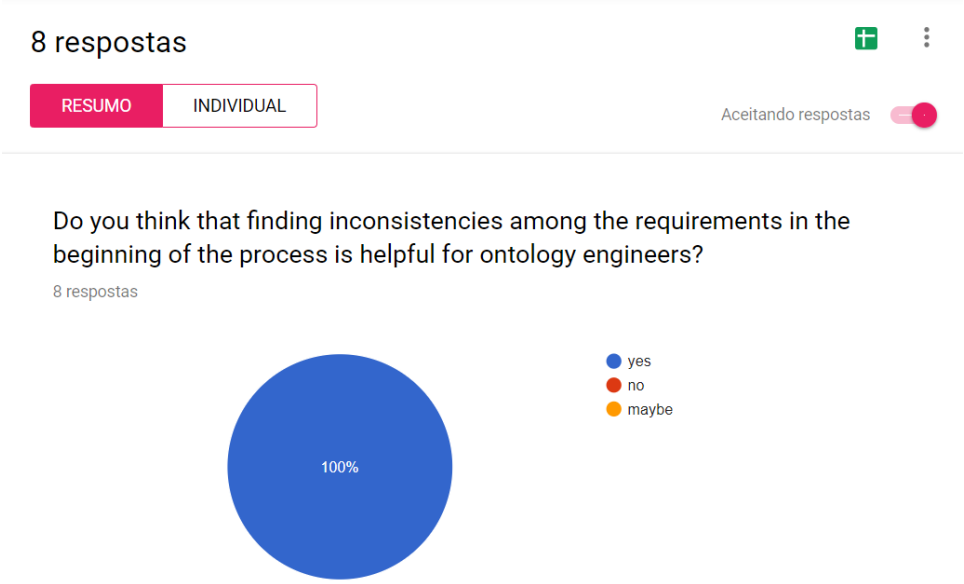


Figura 39 – Experimento: pergunta 1

O resultado para a pergunta "A abordagem ajuda os desenvolvedores de ontologias a poupar tempo e trabalho?" é exibido na Figura 40. 37,5% dos particpantes afirmaram com 100% de certeza que sim e 50% dos participantes afirmaram positivamente com 80% a 90% de certeza.

Na Figura 41 é exibido um gráfico com o resultado para a pergunta "A abordagem ajuda um desenvolvedor de ontologias a construir uma ontologia modular e consistente?". 25% dos participantes afirmaram que sim com 100% de certeza, 37,5% dos participantes

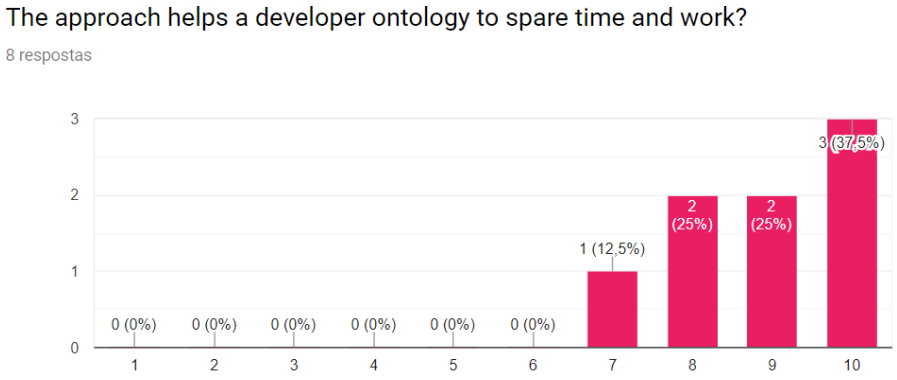


Figura 40 – Experimento: pergunta 2

afirmaram que sim com 90% de certeza, e apenas 12,5% dos participantes afirmaram que sim com 70% de certeza.

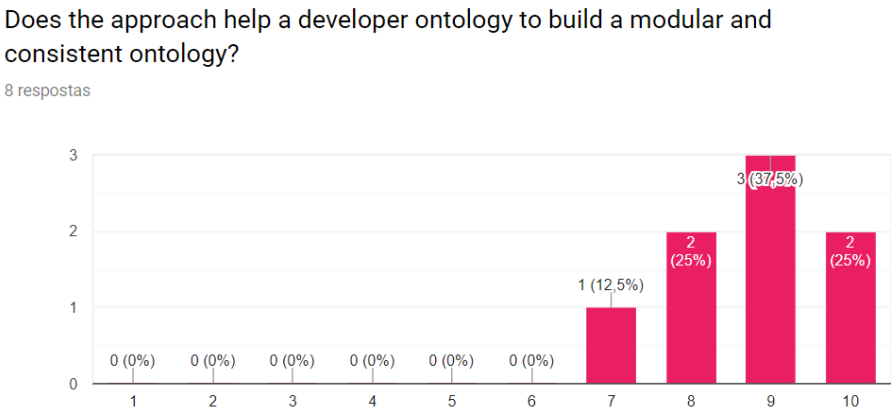


Figura 41 – Experimento: pergunta 3

Na Figura 42 é exibido um gráfico com o resultado para a pergunta “A linguagem natural controlada é fácil de ser utilizada para criação de axiomas?”. 25% dos participantes afirmaram que sim com 100% de certeza, 37,5% dos participantes afirmaram que sim com 90% de certeza, e apenas 12,5% dos participantes afirmaram que sim com 70% de certeza.

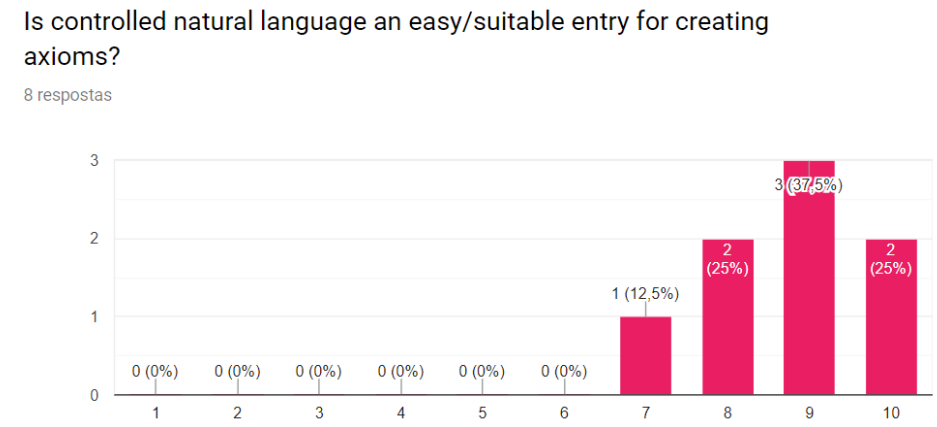


Figura 42 – Experimento: pergunta 4

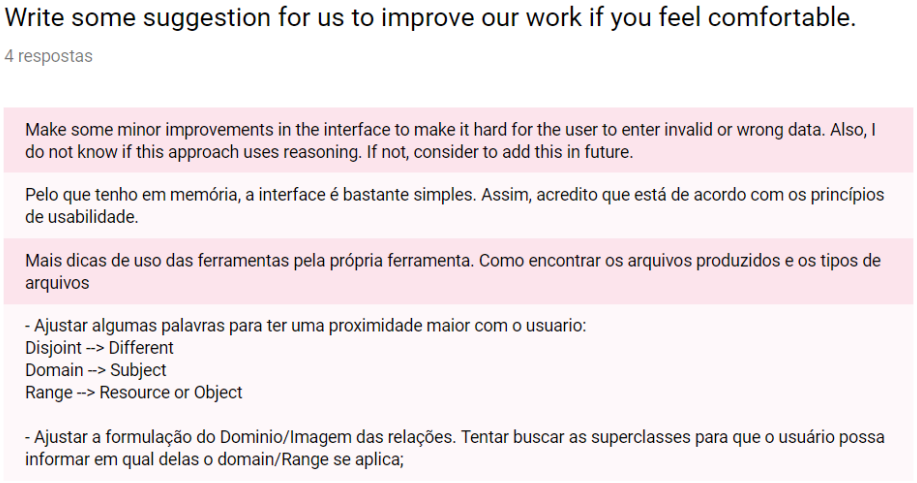


Figura 43 – Experimento: pergunta 5

7.3.4 Métricas

Aplicamos métricas a módulos para analisar se atendia aos critérios de coesão, acoplamento e de completude de satisfação de requisitos.

As métricas foram aplicadas sobre os testes realizados com os desenvolvedores. Utilizamos as sequintes QCs relacionadas a ontologia MadCows:

1. Which food does a dog eat?
   ⇒ Answer: dog is animal that eat bone
   domain of property “eat” is animal

2. Who owns this cat?
   ⇒ Answer: catlover

3. What is sheep?
   ⇒ Answer: sheep is an animal that eat grass

4. What is brain?
   ⇒ Answer: brain is an organ that part of animal
   range of the property “part of” is animal

5. What is vegetarian?
   ⇒ Answer: vegetarian is an animal that does not eat (animal or (part of animal))

6. What is cow?
   ⇒ Answer: cow is vegetarian

7. What is madcow?
   ⇒ Answer: madcow is cow that eat brain and (part of sheep)

A ferramenta gerou, como esperado, sete módulos correspondente a cada QC. Por exemplo, para QC "What is madcow?", a Figura 44 ilustra o fragmento do módulo criado.

Percebe-se que o módulo possui **“What is madcow?”, “What is cow?”, “What is sheep?”, “what is brain?”** e as propriedades **“partOf”** e **“eat”**.

Vamos descrever um exemplo de avaliação das métricas para o módulo “What is a madcow?”.

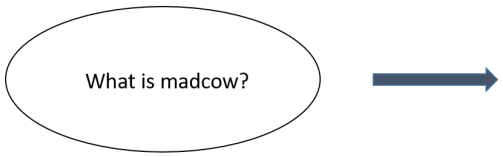
Segundo (ABB‘ES et al., 2012), adotamos as seguintes métricas, para avaliar os módulos:

7.3.4.1 Tamanho do módulo:

Número de classes e propriedades;

Para o módulo “**What is a madcow?**”:

Número de classes: 4 (madcow,sheep,cow,brain)




```
<owl:Class rdf:about="http://www.co-ode.org/ontologies/testont.owl#madcow">
    <rdfs:subClassOf rdf:resource="http://www.co-
ode.org/ontologies/testont.owl#cow"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.co-
ode.org/ontologies/testont.owl#eat"/>
        <owl:someValuesFrom rdf:resource="http://www.co-
ode.org/ontologies/testont.owl#eat"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.co-
ode.org/ontologies/testont.owl#partof"/>
            <owl:someValuesFrom rdf:resource="http://www.co-
ode.org/ontologies/testont.owl#sheep"/>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
    </rdfs:subClassOf>
  </owl:Class>
 <owl:ObjectProperty rdf:about="http://www.co-
ode.org/ontologies/testont.owl#partof">
    <rdfs:range rdf:resource="http://www.co-
ode.org/ontologies/testont.owl#animal"/>
    <rdfs:domain rdf:resource="http://www.co-
ode.org/ontologies/testont.owl#brain"/>
  </owl:ObjectProperty>
....
  <!-- http://www.co-ode.org/ontologies/testont.owl#brain -->
....
 <!-- http://www.co-ode.org/ontologies/testont.owl#sheep -->
```

Figura 44 – MadCow Module

Número de propriedades: 2 (eat, partOf)
Já que, a resposta para essa pergunta é: *madcow* is *cow* that eats *brain* and (part of *sheep*)

7.3.4.2 Coesão do módulo:

Enfatiza a consistência interna de um módulo, a independência do módulo com o mundo exterior. Segundo (ABB'ES et al., 2012), a métrica tem valor entre 0 e 1. Quanto maior a coesão, melhor a modularização. As seguintes métricas são definidas abaixo:

- **Hierarchical Class Chesion(HCC):** O número de ligações hierárquicas entre classes direta ou indiretamente.

$$\text{HCC} = \frac{2*(NdHC+NidHC)}{NC^2-NC}$$

onde NdHC: número de relações diretas de hierarquia entre classes, NidHC: número de relações indiretas de hierarquia entre classes, e NC: número de classes.

$\Rightarrow$ **HCC** $= 1/2^2 - 2 = 1/2 = 0,5$

- **Role Cohesion(RC):** número de ligações hierárquicas direta ou indiretamente entre papéis.

$$\text{RC} = \frac{2*(NdR+NidR)}{NRoles^2-NRoles}$$

onde, NdR: número de papéis diretas entre classes, NidR= número de papéis indiretos entre classes, e NRoles: número de papéis

$\Rightarrow \mathbf{RC} = 2 + 0/2^2 - 2 = 2/4 = 0,5$

- **Object Property Cohesion(OPC):** o número de classes as quais tem associação através de um *object property* em particular(domínio e imagem).

$$\text{OPC} = \frac{2*\sum_{i=1}^{NRoles} NdC(r_i)*NrC(r_i)}{NRoles*(Nc^2-NC)}$$

onde, $NdC(r_i)$: número de classes da ontologia do domínio da relação r_i, $NrC(r_i)$: número de classes da ontologia na imagem da relação r_i, NC: número de classes, e NRoles: número de relações.

$\Rightarrow \mathbf{OPC} = 2 * (1 * 1 + 1 * 1)/2 * (4^2 - 4) = 4/2 * (16 - 4) = 4/12 = 0,3$

- **Coesão** é computada como:

$$\text{Coesão} = \frac{\alpha*HCC+\beta*RC+\sigma*OPC}{\alpha+\beta+\sigma}$$

onde, α,β e σ significam o nível de impacto da coesão de cada tipo de hierarquia de classe, papéis ou object property. Utilizamos $\alpha = \beta = \sigma = 1$, como no artigo

$\Rightarrow$ **Coesão**$=(1 * 0,5 + 1 * 0,5 + 1 * 0,3)/3 = 2,3/3 = 0,7$

Obtivemos uma coesão de 0,7 numa escala de 0 a 1. Como quanto maior, mais próximo de 1, melhor a coesão, consideramos que a coesão do módulo satisfatória.

7.3.4.3 Acoplamento

Acoplamento: envolve aspectos das relações entre os módulos e mede a interdependência entre módulos. Estima a interdependência de diferentes módulos. Quanto menor a interdependência, melhor o resultado o acoplamento. As seguintes métricas são definidas abaixo:

- **Dependência na hierarquia de classes (*Hierarchical class dependency(HCD)*)**: o número de todas as relações de hierarquia de classes direta ou indiretamente com ontologias externas.

$$\mathbf{HCD} = \frac{1}{2} * (\frac{NedHC}{NdHC} + \frac{NeidHC}{NidHC})$$

onde, NedHC: número de dependência de hierarquia de classes direta entre classes locais e classes externas, e NeidHC = de dependência de hierarquia de classes indireta entre classes locais e classes externas.

⇒ HCD= 0,5 * (0/0+0/0)=0

- **Dependência na hierarquia de papéis (*Hierarchical role dependency(HRD)*)**: número de todas as relações de hierarquias de papéis direta ou indireta com ontologias externas.

$$\mathbf{HRD} = \frac{1}{2} * (\frac{NdHR}{NdHR} + \frac{NeidHR}{NidHR})$$

onde, NdHR: número de papéis diretos dependentes entre classes locais e classes externas, e NeidHR: número de papéis indiretos dependentes entre classes locais e classes externas.

⇒ HRD= 0

- **Dependência nos relacionamentos do tipo Object Property (*Object property dependency(OPD)*):** número de papéis que associa classes externas com classes locais.

$$\mathbf{OPD} = \frac{NeRoles}{NRoles}$$

onde, NeRoles: número de todos os papéis que tem uma classes externa no domínio ou imagem deles, NRoles: número de todos os papéis existentes na ontologia.

⇒ OPD= 0

Percebe-se que em relação ao acoplamento, tivemos o valor 0, já que os módulos quando decompostos não apresentam ligação com outros módulos. Pois, se houver uma subQC em módulo A sendo utilizada em outro módulo B, na decomposição a subQC é copiada para cada módulo, evitando ligações externas.

7.3.5 Resumo da aplicação das métricas em outros módulos

A Tabela mostra o resultado para os seguintes módulos da ontologia MadCows e módulos da ontologia Pizza, após a aplicação da decomposição:

1. Pizza: Which are the toppings of margherita pizza?
2. Pizza: Which are the toppings of ice cream?
3. Pizza: Does American have mozzarella topping?

4. Pizza: Is LaReine a pizza or a topping?
5. MadCows: Which food does a dog eat?
6. MadCows: Who owns this cat?
7. MadCows: Where is the heart?
8. MadCows: What is vegetarian?
9. MadCows: What is madcow?

Módulo	Número de Classes	Número de Propriedades	Coesão	Acoplamento
1	3	2	0,7	0
2	2	2	0,8	0
3	3	1	0,6	0
4	2	1	0,6	0
5	3	2	0,8	0
6	1	1	0,4	0
7	1	1	0,4	0
8	3	2	0,7	0
9	4	2	0,7	0

Para todos os módulos que foram aplicado métricas, o acoplamento foi igual a 0, pois estamos tratando com módulos provenientes de uma decomposição, os quais são independentes, não possui ligações com módulos externos.

7.3.5.1 Completude de satisfação de requisitos

Mede se o domínio de interesse é adequadamente coberto. Todas as perguntas, que a ontologia deve ser capaz de responder, podem ser respondidas. Vale ressaltar que a completude neste trabalho está relacionada ao conjunto de QCs inseridas pelo usuário, conjunto que
o usuário julga ser suficiente para checar se as QCs satisfazem o módulo ou ontologia analisado.

Utilizando o CQChecker (ver capítulo 8), descobrimos que todas as QCs satisfazem o módulo ou a ontologia analisada.

7.3.6 Discussão dos experimentos

Com a realização dos experimentos, pudemos constatar o potencial positivo da abordagem proposta para modularização e checagem de satisfatibilidade entre QCs.

Contudo, acreditamos que, por meio de mais testes com desenvolvedores e outras formas de experimentos, poderemos descobrir mais funcionalidades a inserir e encontrar

mais limitações, com o intuito de melhorar, tanto a abordagem, quanto a ferramenta construída.

8 CQCHECKER

O módulo CQChecker ((BEZERRA; SANTANA; FREITAS, 2014))é responsável por verificar a consistência de uma ontologia utilizando QCs. Tal funcionalidade já desenvolvida e publicada, tanto em conferências, como em revista. O objetivo principal desta versão é verificar as questões de competência de uma ontologia, se foram implementadas na ontologia. O principal diferencial, com relação às abordagens mostradas no capítulo 5, é que o CQChecker não considera apenas o ABox da ontologia utilizando consultas SPARQL como modo de verificação, e sim a parte TBox da ontologia, ou seja, usa-se raciocínio em lógica de descrição.

A seguir, uma explanação sobre essa parte já desenvolvida. Basicamente, o usuário digita uma QC e o sistema retorna se a ontologia consegue respondê-la.

O CQChecker é composto por três módulos principais, exibidos na Figura 45. Cada um deles trata um tipo de questão de competência. Na ferramenta, são considerados os tipos abaixo:

- **Tipo 1: QCs que tratam sobre classes e suas relações.** Neste caso, a resposta consiste em uma lista de classes e instâncias. Por exemplo, para a QC "*Which are the topping of the cheesey pizza?*", a classe "*CheeseTopping*" seria retornada de acordo com a ontologia de Pizza disponível no site do Protégé;

- **Tipo 2: Problemas de decisão expressadas como QCs.** Neste tipo, a resposta será verdadeiro ou falso. Dependendo da QC, pode ser necessário um processo de inferência para respondê-la e, assim, caso a QC seja verdadeira, é mostradoo motivo;

- **Tipo 3: QCs que lidam com instâncias da ontologia.** Por exemplo, "*Which are the possible countries of a Pizza?*" obtemos a resposta "*France*, *Italy*, *England*, *Germany*, *America*".

Já que estamos lidando com linguagem natural controlada, temos um conjunto de padrões de QCs ao qual o sistema pode responder. A Figura 46 ilustra parte desses padrões.

Para implementação, foram utilizados a OWL API de Manchester((HORRIDGE; BECHHOFER, 2011)), Wordnet((MILLER, 1995)), algoritmo de similaridade de Levenshtein disponível na Similarity API[1] e o raciocinador Pellet((SIRIN et al., 2007)).

[1] https://github.com/tdebatty/java-string-similarity

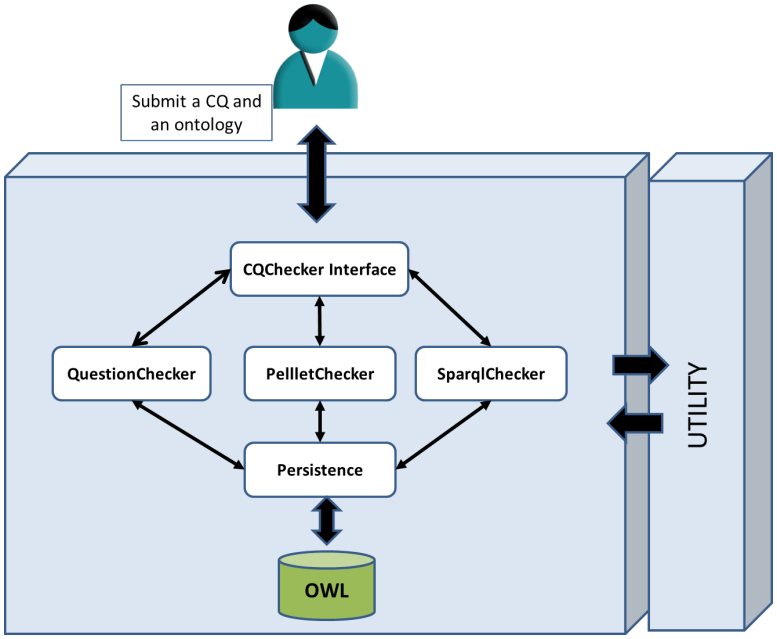


Figura 45 – Arquitetura do CQChecker

Pattern	Example	Ontology entities found in CQ
What + <property> +<class>?	What is the spiciness of chicken topping?	<property> = *hasSpiciness* <class> = *ChickenTopping*
How many + <property> + <class>?	How many toppings does the interesting pizza have?	<property> = *hasTopping* <class> = *InterestingPizza*
From which + <property> + <class>?	From which nation is American pizza?	<property> = *hasCountryOfOrigin* <class> = *American*
Is + <class>+ a +<class>?	Is pizza a food?	<class> = *Pizza* <class> = *Food*
Is + <individual>+ a +<class>?	Is Marguerita a vegetarian pizza?	<individual> = *Marguerita* <class> = *VegetarianPizza*
Which + type + <class>?	Which are the types of Pizza?	<class> = *Pizza*
Does <class> + <property> + <class>?	Does American pizza have mozzarella topping?	<class> = *American* <class> = *MozarellaTopping*
Which + disjoint <class>?	Which are the disjoint classes of Vegetarian Pizza?	<class> = *VegetarianPizza*
Are + <class> + <class> disjoint?	Are vegetarian pizza and non-vegetarian pizza disjoint?	<class> = *VegetarianPizza* <class> = *NonVegetarianPizza*
Which + <class> + <property> + <class> + not + <property> + <class>?	Which are the pizzas that have mozzarella topping but not have meat topping?	<class> = *Pizza* <property> = *hasTopping* <class> = *MozzarellaTopping* <class> = *MeatTopping*

Figura 46 – Parte dos padrões linguísticos aceitos no CQChecker

8.1 QUESTIONCHECKER

Esse módulo é o mais complexo e trabalha com o tipo de QC 1. Para isso, é utilizado o algoritmo apresentado na Figura 47. Basicamente, ele recebe uma QC, divide em tokens e tenta encontrar os conceitos e as relações da ontologia descritos em OWL DL, aos quais a QC se refere. Considere a QC "Quais são as coberturas do CheeseyPizza?". Com isso, queremos verificar se há qualquer classe que iria fornecer uma instância para a imagem da classe *CheeseyPizza*, através da relação *hasTopping*, que é um *ToppingPizza*. Para que isso seja alcançado, primeiro é preciso procurar uma relação, que pode ser um verbo ou um substantivo na questão de competência. Em seguida, olhar para o conceito, que tem uma imagem que é uma *ToppingPizza* na relação, no caso, *CheeseyPizza*. O algoritmo irá realizar os seguintes passos:

1. Dividir a QC em uma lista de tokens
    - "Which", "are", "the", "Toppings", "of", "the", "Cheesey" e "Pizza"
2. Encontrar na QC, a relação(propriedade) principal, à qual a classe principal se refere
    - No caso, a principal relação é "hasTopping", pois o único token mais similar na QC é "toppings"
    - Como a relação já foi encontrada, esse token é deletado da lista de tokens
3. Encontrar, entre os tokens, a classe principal
    - A classe é *CheeseyPizza*, porque cheesey e pizza são os únicos tokens que sobraram que, juntos, formam a classe que existe na ontologia *CheeseyPizza*
4. Obter da ontologia os axiomas que contêm a classe e a relação principal
    - Neste caso, o sistema retorna o axioma em OWL
      ObjectSomeValuesFrom(
      <<http://www.co-ode.org/ontologies/pizza/pizza.owl#hasTopping>>
      <<http://www.co-ode.org/ontologies/pizza/pizza.owl#CheeseTopping>>)
5. Selecione os axiomas que contêm o conceito e a relação principal
    - No caso, o axioma do passo anterior.
6. Identificar o tipo de axioma
    - No exemplo, uma restrição existencial, ou seja, $\exists P.C$.
7. Identificar e retornar os conceitos do axioma encontrado.
    - No caso, CheeseTopping

O passo 6 do algoritmo pode ser detalhado a seguir:

1. O algoritmo busca o conceito principal em comparação com os nomes de classe dos conceitos da ontologia, ignorando letras maiúsculas e minúsculas;

2. Primeiro ele verifica se a última palavra é uma classe. Se sim, ele pergunta ao usuário se essa palavra é o principal conceito da QC. Se o usuário confirmar, então a última palavra será a *MainConceptToken*. Caso contrário, ele vai tentar o próximo passo;

3. Em seguida, ele verifica se a penúltima palavra é uma classe. Se sim, ele pergunta ao usuário se essa palavra é o principal conceito da QC. Se o usuário confirmar, essa penúltima palavra é considerada a *MainConceptToken*. Caso contrário, ele vai tentar o próximo passo;

4. É testado se os dois últimos e os três últimos tokens da QC em conjunto consistem em uma classe da ontologia. Se sim, ele pergunta ao usuário se essa palavra é o principal conceito da QC. Se o usuário confirmar, os dois ou três últimos tokens serão considerados como o *MainConceptToken* da QC;

5. Se, mesmo após os testes anteriores, o principal conceito não for encontrado, então a API *WordNet* é utilizada para obter os sinônimos das últimas palavras e testar se um deles é uma classe. Se sim, essa classe será considerada o *MainConceptToken*.

Suponha que o usuário digite uma QC, mas não saiba bem os nomes das classes e propriedades. Nesse caso, ele pode utilizar sinônimos no lugar. Por exemplo, considerando a QC "From which nation is the American pizza?", a palavra "*nation*" não corresponde a nenhuma classe existente na ontologia. No caso, o QuestionChecker obtém sinônimos da palavra "*nation*" e testa se um dos sinônimos existentes corresponde a alguma propriedade na ontologia. No exemplo citado, ele achará o sinônimo "*country*" similar à propriedade "*hasCountryOfOrigin*" na ontologia. Então, será perguntado ao usuário se é essa a propriedade; e assim, será executado o resto do algoritmo utilizando a propriedade "*hasCountryOfOrigin*".

Question Checker

Inputs: CQ_{NL} (Competency questions in controlled natural language),

O = set of axioms a_i defined over the triple (N_C, N_R, N_O), where

$-N_C$ is the set of *concept names*,

- N_R is the set of *role or property names*,

- N_I the set of *instances* of N_C and N_R:

Output: $\propto \mid O \models \propto (CQ_{DL})$, where CQ_{DL} is a competency question written in DL

1. TokenSet := {set of tokens | token extracted from CQ_{NL}}
2. MainRoletToken := token[1]
3. **For** i=1 **to** size(TokenSet)
    a. **For** j=1 **to** size(N_R)
        i. **If** token[i] = r[j] ∈ N_R **then**
            1. MainRoleToken := r[j]
            2. **Break**
        ii. **Endif**
    b. **Endfor**
4. **Endfor**
5. MainConceptToken := token[1]
6. **For** i=1 **to** size(TokenSet)
    a. **For** j=1 **to** size(N_C)
        i. **If** token[i] = C[j] ∈ N_C **then**
            1. MainConceptToken := C[j]
            2. **Break**
        ii. **Endif**
    b. **Endfor**
7. **Endfor**
8. AxiomSet = {a' ∈ a_i | MainConceptToken $\in_{lexically}$ a and MainRoleToken $\in_{lexically}$ a}
9. $CQ_{DL} := \wedge_{i=1}^{n} a'_i \in$ AxiomSet
10. **return** $\propto \mid O \models \propto (CQ_{DL})$

Figura 47 – Algoritmo do QuestionChecker ((BEZERRA; FREITAS; SANTANA, 2013))

8.2 RESULTADOS

Fizemos três rodadas de testes, com ontologias diferentes. Para cada rodada, dez QCs foram usadas. As ontologias utilizadas estão disponíveis no site do Protégé[2]. Nos primeiros testes, as QCs foram criadas pela equipe de desenvolvimento. Acreditamos que as QCs escolhidas são representativas para este primeiro momento. Num futuro próximo, pretendemos criar testes robustos com grandes ontologias, incluindo testes realizados por pessoas de fora da equipe de desenvolvimento. As QCs estão relacionadas a hierarquias de classes, indivíduos, classes disjuntas, intersecção e união de classes; classes equivalentes, quantificação universal e existencial; Restrição de "valor" e restrição de cardinalidade.

Para a ontologia de Pizza, os resultados são apresentados na Tabela 4.
Para mais detalhes sobre o CQChecker consultar o artigo (BEZERRA; SANTANA; FREITAS, 2014).

[2] http://protege.stanford.edu/download/ontologies.html

Tabela 4 – Teste com a ontologia de Pizza

CQ	**Answer**	**Correct?**
What is the spiciness of a PeperoniSausageTopping?	Medium	yes
Which are the Toppings of a VegetarianPizza?	NOT (MeatTopping and FishTopping)	yes
Which are the Toppings of the CheeseyPizza?	CheeseTopping	yes
What is the Base of ThinAndCrispyPizza?	ThinAndCrispyBase	yes
Are ChickenTopping disjoint of HamTopping?	yes	yes
What is the countryOfOrigin of MozzarellaTopping?	Italy	yes
Which are the types of CheeseTopping?	CheeseyVegetableTopping MozzarellaTopping GorgonzolaTopping GoatsCheeseTopping ParmesanTopping FourCheesesTopping	yes
How many toppings have the InterestingPizza?	3	yes
Which are the members of the Country?	France,England,Italy,Germany,America	yes
Which are the classes disjoint of VegetarianPizza?	NonVegetarianPizza	yes

A necessidade de fazer buscas e obter resultados mais precisos de consultas na Web levou à criação da web semântica e de seus padrões. Com o crescimento da web semântica e outras áreas que fazem uso de ontologias, é cada vez mais crucial o reuso de ontologias, visando alcançar escalabilidade, gerenciamento e raciocínio. A modularização de ontologias surge como uma proposta de resolver esse problema, seguindo ideia de dividir para conquistar.

Além disso, assim como na Engenharia de Software, a Engenharia de Ontologias possui lacunas a serem resolvidas. Os requisitos de ontologias são conhecidos como QCs, os quais precisam ser definidos na especificação e validação de requisitos. Nesta tese, propomos pioneiramente a modularização e checagem de satisfatibilidade lógica das QCs já no início do processo, de forma semiautomática, trazendo uma série de benefícios para o desenvolvimento. A seguir, vamos descrever as contribuições alcançadas, estabelecidas no capítulo 1, fazendo uma breve discussão sobre o que foi atingido e as perspectivas para trabalhos futuros.

9.1 CONTRIBUIÇÕES

9.1.1 Ferramenta que verifique se uma ontologia ou módulo satisfaz os requisitos, ou seja, as QCs em LD

Na etapa de revisão bibliográfica, notamos que existia uma lacuna no que diz respeito à checagem da satisfatibilidade de uma ontologia utilizando suas QCs. Boa parte dos trabalhos investigados eram realizados manualmente, não tinham auxílio de um software e, quando tinham, verificava-se apenas o nível assercional (ABox) das ontologias, utlizando a linguagem SPARQL ((ANGLES; GUTIERREZ, 2008)).

Para resolver o problema de checagem automática em nível terminológico, criamos o CQChecker descrito no capítulo 8. O CQChecker utiliza QCs de uma ontologia em linguagem natural controlada para checar se as questões de competência satisfazem essa ontologia. Tal checagem é realizada no final do desenvolvimento da ontologia.

Este trabalho nos auxiliou a estabelecer os objetivos seguintes e principais da proposta. Com o CQChecker, conseguimos delinear melhor o problema principal abordado nesta tese, que consiste na modularização e checagem de satisfatibilidade de QCs no início do desenvolvimento de uma ontologia.

9.1.2 Abordagem para modularização de ontologias baseada em QCs e verificação de inconsistência lógica entre duas ou mais CQs

A contribuição principal da tese é a modularização da ontologia em QCs no início do desenvolvimento. Com isso, é possível checar se existem insatisfatibilidades entre as próprias QCs, algo nunca explorado até então.

Utilizando essa abordagem, o desenvolvedor, além de utilizar as QCs para caracterizar o domínio da ontologia, antes mesmo de construí-la, pode descobrir prováveis inconsistências que seriam diagnosticadas em fases mais adiantadas do desenvolvimento, além de obter uma ontologia modularizada a partir de QCs. As vantagens dessa abordagem são explicitadas abaixo:

- Descobrir insatisfatilidades nas próprias QCs, ou seja, nos requisitos da ontolo gia, no início do desenvolvimento, com isso, economizando tempo e facilitando a manutenção;
- Ao aplicar a abordagem, o desenvolvedor terá uma ontologia construída conforme as QCs, pois é a partir delas que os axiomas terminológicos são inseridos na ontologia;
- Criação de módulos a partir das QCs, onde cada QC corresponde a um módulo;
- O rastreamento de requisitos passa a ser diretamente identificado, pois cada QC se transforma em um módulo e, portanto, o requisito está satisfeito no módulo correspondente à QC;
- Opção de fazer composição, decomposição e extração de módulos, respectivamente, a partir de QCs.

9.1.3 Criação da ferramenta que implemente a abordagem

Com o objetivo de validar a abordagem proposta, criamos uma ferramenta chamada CQSetConsistencyChecker.Com ela, é possível verificar se a ontologia possui conjuntos de QCs insatisfatíveis entre si. A ferramenta gera ontologias em OWL $\mathcal{ALCN}$.

A ferramenta CQSetConsistencyChecker suporta:

- Composição, decomposição e extração de módulos baseados em QCs;
- Checagem de satisfatibilidade entre os módulos. Se houver insatisfatibilidade, são exibidas as QCs envolvidas com a causa;
- Geração de uma ontologia OWL, facilitando assim o processo de construção;
- Visualização gráfica dos módulos, destacando, se houver, as QCs conflitantes;
- Exportação dos dados de todo o processo de criação da ontologia modularizada.

Além disso, como descrito no capítulo 7, foi realizado um experimento com desenvolvedores externos de ontologias, com intuito de avaliarem a abordagem através da ferramenta. Os participantes utilizaram a ferramenta e responderam a um questionário onde foi constatado que:

- 100% acreditam que encontrar inconsistências entre os requisitos (QCs) no início do processo é útil para os engenheiros de ontologia;
- 70% a 100% acreditam que a abordagem economiza tempo e trabalho;
- 90% acreditam que ajuda o desenvolvedor a criar uma ontologia modular e consistente;
- 90% acreditam que o padrão de linguagem natural controlada adotado no trabalho é fácil para definição de axiomas dentro da ferramenta.

Com esses experimentos, chegamos à conclusão de que a abordagem atende à proposta, ou seja, tem potencial de melhorar o desenvolvimento de ontologias.

9.2 ARTIGOS PUBLICADOS

Até o momento, foram publicados os seguintes trabalhos:

- Bezerra, Camila; FREITAS, F. L. G. . Verifying Description Logic Ontologies based on Competency Questions and Unit Testing. In: ONTOBRAS - Seminário de Pesquisa em Ontologias do Brasil, 2017, Brasilia. ONTOBRAS 2017- Seminário de Pesquisa em Ontologias do Brasil, 2017.
- C. Bezerra, F. Santana, and F. Freitas, “Cqchecker: A tool to check ontologies in owl-dl using competency questions written in controlled natural language. Learning and Nonlinear Models, v.12, p. 115-129, 2014.
- Bezerra, Camila; FREITAS, F. L. G. ; Santana, F . CQChecker: A Tool to Check the Satisfaction of Description Logic Competency Questions on Ontologies. In: X Encontro Nacional de Inteligência Artificial e Computacional (ENIAC), 2013, Fortaleza. X Encontro Nacional de Inteligência Artificial e Computacional (ENIAC), 2013.
- C. Bezerra, F. Freitas, and F. Santana, “Evaluating ontologies with competency questions”, in 2013 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology, Atlanta, Georgia, USA, 17-20 November 2013, Workshop Proceedings, 2013, pp. 284–285.

9.3 LIMITAÇÕES

Discorreremos acerca das limitações da abordagem e da ferramenta:

9.3.1 Limitações da Abordagem

- Apesar de a visão de projeto de LD levar em conta que os axiomas estão espalhados sobre as classes ou pela web ou pela base de conhecimento, assumimos neste trabalho que isso não ocorre;
- Expandir para outras linguagens LDs.

9.3.2 Limitações da ferramenta

- Processamento de linguagem natural limitado. Além de estarmos utilizando um conjunto de padrões que podem ser aumentados, eles também podem ser enriquecidos com mais recursos linguísticos (e.g. ferramentas semânticas, como o FrameNet ((BAKER; FILLMORE; LOWE, 1998)) e o Semafor (CHEN et al., 2010), além de técnicas de PLN relacionadas a eles;
- A ferramenta só consegue tratar certos padrões de QCs em $\mathcal{LD}$ $\mathcal{ALCN}$, e o número e formato de padrões naturalmente podem ser aumentados;
- As subQCs são apenas questões do tipo "What", ou seja, todas as QCs são do tipo definitoriais;
- Para uso em ambientes profissionais, é preciso enriquecer a ferramenta a fim de abarcar mais padrões;
- A ferramenta não fornece a possibilidade de editar o processo de modularização em tempo real.

9.4 TRABALHOS FUTUROS

Diante dos objetivos alcançados, pretendemos que o resultado final do trabalho seja publicado, para que a proposta ganhe mais visibilidade e usuários. Para tratar as limitações, são necessários, entre outros, atacar os seguintes problemas:

- Resolver as limitações existentes no processamento de linguagem natural controlada;
- Estender para outras DLs, além de $\mathcal{LD}$ $\mathcal{ALCN}$. Por exemplo, passar para $\mathcal{LD}$ $\mathcal{SROIQ}$;
- Melhorar a usabilidade da ferramenta, para que ela possa ser utilizada em larga escala;

- Realizar mais experimentos.

REFERÊNCIAS

ABB'ES, S. B.; SCHEUERMANN, A.; MEILENDER, T.; D'AQUIN, M. Characterizing modular ontologies. In: *International Conference on Formal Ontologies in Information Systems(FOIS)*. [S.l.: s.n.], 2012.

ALDOSARI, B.; ALANAZI, A.; HOUSEH, M. Pitfalls of ontology in medicine. *Studies in health technology and informatics*, v. 238, p. 15–18, 01 2017.

Allen, E. B.; Khoshgoftaar, T. M.; Chen, Y. Measuring coupling and cohesion of software modules: an information-theory approach. In: *Proceedings Seventh International Software Metrics Symposium*. [S.l.: s.n.], 2001. p. 124–134. ISSN 1530-1435.

ALLEN, G.; OWENS, M. *The Definitive Guide to SQLite*. 2nd. ed. Berkely, CA, USA: Apress, 2010. ISBN 1430232250, 9781430232254.

ANGLES, R.; GUTIERREZ, C. The expressive power of sparql. In: SHETH, A.; STAAB, S.; DEAN, M.; PAOLUCCI, M.; MAYNARD, D.; FININ, T.; THIRUNARAYAN, K. (Ed.). *The Semantic Web - ISWC 2008*. Springer Berlin Heidelberg, 2008, (Lecture Notes in Computer Science, v. 5318). p. 114–129. ISBN 978-3-540-88563-4. Disponível em: <http://dx.doi.org/10.1007/978-3-540-88564-1_8>.

ANNAMALAI, M.; SANIP, Z. Natural language support for competency evaluation of web-ontologies. *Journal of IT in Asia*, v. 3, 2010.

BAADER, F.; BRANDT, S.; LUTZ, C. Pushing the EL Envelope. In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005. (IJCAI'05), p. 364–369. Disponível em: <http://dl.acm.org/citation.cfm?id=1642293.1642351>.

BAADER, F.; CALVANESE, D.; MCGUINNESS, D. L.; NARDI, D.; PATEL-SCHNEIDER, P. F. *The Description Logic Handbook: Theory, Implementation and Applications*. 2nd. ed. New York, NY, USA: Cambridge University Press, 2010. ISBN 0521150116, 9780521150118.

BAADER, F.; HORROCKS, I.; SATTLER, U. Description Logics. In: van Harmelen, F.; LIFSCHITZ, V.; PORTER, B. (Ed.). *Handbook of Knowledge Representation*. Elsevier, 2008. cap. 3, p. 135–180. Disponível em: <download/2007/BaHS07a.pdf>.

BAKER, C. F.; FILLMORE, C. J.; LOWE, J. B. The berkeley framenet project. In: *Proceedings of the 17th International Conference on Computational Linguistics*. Stroudsburg, PA, USA: Association for Computational Linguistics, 1998. (COLING '98, v. 1), p. 86–90. Disponível em: <https://doi.org/10.3115/980451.980860>.

BAO, J.; VOUTSADAKIS, G.; SLUTZKI, G.; HONAVAR, V. Modular ontologies. In: STUCKENSCHMIDT, H.; PARENT, C.; SPACCAPIETRA, S. (Ed.). Berlin, Heidelberg: Springer-Verlag, 2009. cap. Package-Based Description Logics, p. 349–371. ISBN 978-3-642-01906-7. Disponível em: <http://dx.doi.org/10.1007/978-3-642-01907-4_16>.

BARD, J.; RHEE, S. Y. Ontologies in biology: Design, applications and future challenges. *Nature reviews. Genetics*, v. 5, p. 213–22, 04 2004.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The Semantic Web. *Scientific American*, v. 284, p. 34–43, 2001.

BEZERRA, C.; FREITAS, F.; SANTANA, F. Evaluating ontologies with competency questions. In: *IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology, Atlanta, Georgia, USA, 17-20 November 2013, Workshop Proceedings*. [s.n.], 2013. p. 284–285. Disponível em: <http://dx.doi.org/10.1109/WI-IAT.2013.199>.

BEZERRA, C.; FREITAS, F. L. G. de; ZIMMERMANN, A.; EUZENAT, J. Modonto: A tool for modularizing ontologies. In: *Proceedings of the 3rd Workshop on Ontologies and their Applications, Salvador, Bahia, Brazil, October 26, 2008*. [s.n.], 2008. Disponível em: <http://ceur-ws.org/Vol-427/paper3.pdf>.

BEZERRA, C.; SANTANA, F.; FREITAS, F. Cqchecker: A tool to check ontologies in owl-dl using competency questions written in controlled natural language: A tool to check ontologies in owl-dl using competency questions written in controlled natural language. *Learning & Nonlinear Models*, SBRN, v. 12, n. 2, p. 115–129, 2014.

BLOMQVIST, E.; SEPOUR, A. S.; PRESUTTI, V. Ontology testing - methodology and tool. In: TEIJE, A. ten; VÖLKER, J.; HANDSCHUH, S.; STUCKENSCHMIDT, H.; D'ACQUIN, M.; NIKOLOV, A.; AUSSENAC-GILLES, N.; HERNANDEZ, N. (Ed.). *Knowledge Engineering and Knowledge Management: 18th International Conference, EKAW 2012, Galway City, Ireland*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 216–226. ISBN 978-3-642-33876-2. Disponível em: <http://dx.doi.org/10.1007/978-3-642-33876-2_20>.

BOHM, G. Industrial software metrics: A top-ten list. *IEEE Software*, v. 5, p. 84–85, 1987.

BRESCIANI, P.; PERINI, A.; GIORGINI, P.; GIUNCHIGLIA, F.; MYLOPOULOS, J. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, Kluwer Academic Publishers, Hingham, MA, USA, v. 8, n. 3, p. 203–236, maio 2004. ISSN 1387-2532. Disponível em: <http://dx.doi.org/10.1023/B:AGNT.0000018806.20944.ef>.

BROCHHAUSEN, M.; FRANSSON, M. N.; KANASKAR, N. V.; ERIKSSON, M.; MERINO-MARTINEZ, R.; HALL, R. A.; NORLIN, L.; KJELLQVIST, S.; HORTLUND, M.; TOPALOGLU, U.; HOGAN, W. R.; LITTON, J.-E. Developing a semantically rich ontology for the biobank-administration domain. *Journal of Biomedical Semantics*, v. 4, n. 1, p. 23, Oct 2013. ISSN 2041-1480. Disponível em: <https://doi.org/10.1186/2041-1480-4-23>.

CAMPBELL, D. T.; STANLEY, J. *Experimental and Quasi-Experimental Designs for Research*. [S.l.]: Cengage Learning, 1963.

CAMPBELL, K. The metaphysic of abstract particulars. *Midwest Studies in Philosophy*, 1981.

CHEN, D.; SCHNEIDER, N.; DAS, D.; SMITH, N. A. SEMAFOR: frame argument resolution with log-linear models. In: *Proceedings of the 5th International Workshop on Semantic Evaluation, SemEval@ACL 2010, Uppsala University,*

Uppsala, Sweden, July 15-16, 2010. [s.n.], 2010. p. 264–267. Disponível em: <http://aclweb.org/anthology/S/S10/S10-1059.pdf>.

CORAZZON, R. *Descriptive and Formal Ontology: A Resource Guide to Contemporary Research*. 2016. Https://www.ontology.co/.

Cuenca Grau, B.; HORROCKS, I.; KAZAKOV, Y.; SATTLER, U. Modular reuse of ontologies: Theory and practice. *J. of Artificial Intelligence Research (JAIR)*, v. 31, p. 273–318, 2008. Disponível em: <http://www.jair.org/media/2375/live-2375-3703-jair.pdf>.

D'AQUIN, M.; SCHLICHT, A.; STUCKENSCHMIDT, H.; SABOU, M. Ontology modularization for knowledge selection: Experiments and evaluations. In: *Database and Expert Systems Applications*. [S.l.: s.n.], 2007. p. 874–883.

DAVIES, J.; STUDER, R.; WARREN, P. (Ed.). *Semantic Web Technologies: Trends and Research in Ontology-based Systems*. [S.l.]: Wiley, 2006.

DAVIS, A. M. *Software Requirements: Objects, Functions, and States*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993. ISBN 0-13-805763-X.

DENNIS, M.; van Deemter, K.; DELL'AGLIO, D.; PAN, J. Computing authoring tests from competency questions: Experimental validation. In: D'AMATO, C.; FERNANDEZ, M.; TAMMA, V.; LECUE, F.; CUDRÉ-MAUROUX, P.; SEQUEDA, J.; LANGE, C.; HEFLIN, J. (Ed.). *The Semantic Web, ISWC 2017*. Germany: Springer Verlag, 2017. (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)), p. 243–259. ISBN 978-3-319-68287-7.

FARQUHAR, A.; FIKES, R.; RICE, J. The ontolingua server: a tool for collaborative ontology construction. *International Journal of Human-Computer Studies*, v. 46, n. 6, p. 707 – 727, 1997. ISSN 1071-5819. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1071581996901214>.

FERNANDES, P. C. B.; GUIZZARDI, R. S. S.; GUIZZARDI, G. Using goal modeling to capture competency questions in ontology-based systems. *JIDM*, v. 2, n. 3, p. 527–540, 2011. Disponível em: <http://seer.lcc.ufmg.br/index.php/jidm/article/view/145>.

FERNÁNDEZ-LÓPEZ, M.; GÓMEZ-PÉREZ, A.; JURISTO, N. Methontology: From ontological art towards ontological engineering. In: *Proceedings of the Ontological Engineering AAAI-97 Spring Symposium Series*. American Asociation for Artificial Intelligence, 1997. Ontology Engineering Group ? OEG. Disponível em: <http://oa.upm.es/5484/>.

FIKES, R.; KEHLER, T. The role of frame-based representation in reasoning. *Commun. ACM*, ACM, New York, NY, USA, v. 28, n. 9, p. 904–920, set. 1985. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/4284.4285>.

FOX, M. S.; GRÜNINGER, M. Ontologies for enterprise integration. In: *CoopIS*. [S.l.: s.n.], 1994. p. 82–89.

FREY, K.; ACCOMAZZI, A. The unified astronomy thesaurus: Semantic metadata for astronomy and astrophysics. *CoRR*, abs/1801.01021, 2018. Disponível em: <http://arxiv.org/abs/1801.01021>.

GARCÍA-RAMOS, S.; OTERO, A.; FERNÁNDEZ-LÓPEZ, M. Ontologytest: A tool to evaluate ontologies through tests defined by the user. In: OMATU, S.; ROCHA, M. P.; BRAVO, J.; FERNÁNDEZ, F.; CORCHADO, E.; BUSTILLO, A.; CORCHADO, J. M. (Ed.). *10th International Work-Conference on Artificial Neural Networks, IWANN 2009 Workshops, Salamanca, Spain*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 91–98. ISBN 978-3-642-02481-8. Disponível em: <http://dx.doi.org/10.1007/978-3-642-02481-8_13>.

GATENS, W.; KONEV, B.; WOLTER, F. Lower and upper approximations for depleting modules of description logic ontologies. In: *Informal Proceedings of the 27th International Workshop on Description Logics, Vienna, Austria, July 17-20, 2014*. [s.n.], 2014. p. 181–184. Disponível em: <http://ceur-ws.org/Vol-1193/paper_15.pdf>.

GOMEZ-PEREZ, A.; CORCHO, O.; FERNANDEZ-LOPEZ, M. *Ontological Engineering : with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. First Edition (Advanced Information and Knowledge Processing)*. Springer, 2004. Hardcover. ISBN 1852335513. Disponível em: <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/1852335513>.

GORSKI, E.; COELHO, I. VariaÃ§Ã£o linguÃstica e ensino de gramÃ¡tica. *Working papers em Linguistica*, v. 10, n. 1, p. 73–91, 2010. ISSN 1984–8420. Disponível em: <https://periodicos.ufsc.br/index.php/workingpapers/article/view/1984-8420.2009v10n1p73>.

GRAU, B. C.; HORROCKS, I.; MOTIK, B.; PARSIA, B.; PATEL-SCHNEIDER, P.; SATTLER, U. Owl 2: The next step for owl. *Web Semantic*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 6, n. 4, p. 309–322, nov. 2008. ISSN 1570-8268. Disponível em: <http://dx.doi.org/10.1016/j.websem.2008.05.001>.

GRAU, B. C.; PARSIA, B.; SIRIN, E. Ontology integration using epsilon-connections. In: *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*. [s.n.], 2009. p. 293–320. Disponível em: <http://dx.doi.org/10.1007/978-3-642-01907-4_14>.

GRAU, B. C.; PARSIA, B.; SIRIN, E.; KALYANPUR, A. Modularizing owl ontologies. In: *Proceedings of the 4th International Semantic Web Conference (Poster Track)*. [S.l.: s.n.], 2005.

GRÜNINGER, M.; FOX, M. Methodology for the design and evaluation of ontologies. In: *IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing*. [S.l.: s.n.], 1995.

GUEDES, A. L. P. *HIPERGRAFOS DIRECIONADOS*. Tese (Doutorado) — UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, 2001.

HAASE, P.; LEWEN, H.; STUDER, R.; TRAN, D. T.; ERDMANN, M.; D'AQUIN, M.; MOTTA, E. The neon ontology engineering toolkit. In: *WWW 2008 Developers Track*. [S.l.: s.n.], 2008.

HOFER, P.; NEURURER, S. B.; HAUFFE, H.; INSAM, T.; ZEILNER, A.; GÖBEL, G. Semi-automated evaluation of biomedical ontologies for the biobanking domain based on competency questions. In: *Health Informatics Meets eHealth - Innovative Health Perspectives: Personalized Health, eHealth 2015, Vienna, Austria, June 2015*. [s.n.], 2015. p. 65–72. Disponível em: <https://doi.org/10.3233/978-1-61499-524-1-65>.

HOFMANN, H. F.; LEHNER, F. Requirements engineering as a success factor in software projects. *IEEE Softw.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 18, n. 4, p. 58–66, jul. 2001. ISSN 0740-7459. Disponível em: <http://dx.doi.org/10.1109/MS.2001.936219>.

HORRIDGE, M.; BECHHOFER, S. The owl api: A java api for owl ontologies. *Semantic Web*, v. 2, n. 1, p. 11–21, 2011.

IEEE. *IEEE Recomenend Practice for Software Requirements Specifications*. [S.l.], 1993.

IEEE. *IEEE Guide for Software Quality Assurance Planning*. [S.l.], 1995.

INDURKHYA, N.; DAMERAU, F. J. *Handbook of Natural Language Processing*. 2nd. ed. [S.l.]: Chapman & Hall/CRC, 2010. ISBN 1420085921, 9781420085921.

KALYANPUR, A.; PARSIA, B.; SIRIN, E.; HENDLER, J. Debugging unsatisfiable classes in owl ontologies. *Web Semant.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 3, n. 4, p. 268–293, dez. 2005. ISSN 1570-8268. Disponível em: <http://dx.doi.org/10.1016/j.websem.2005.09.005>.

KONEV, B.; LUTZ, C.; WALTHER, D.; WOLTER, F. Formal properties of modularisation. In: STUCKENSCHMIDT, H.; PARENT, C.; SPACCAPIETRA, S. (Ed.). *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*. Berlin, Heidelberg: Springer-Verlag, 2009. cap. Formal Properties of Modularisation, p. 25–66. ISBN 978-3-642-01906-7. Disponível em: <http://dx.doi.org/10.1007/978-3-642-01907-4_3>.

KROTZSCH, M.; SIMANCIK, F.; HORROCKS, I. A description logic primer. *CoRR*, abs/1201.4089, 2012.

LENAT, D. B.; GUHA, R. V. *Building Large Knowledge-Based Systems; Representation and Inference in the Cyc Project*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 0201517523.

MALHEIROS, Y.; FREITAS, F. A method to develop description logic ontologies iteratively based on competency questions: an implementation. In: *ONTOBRAS*. [S.l.: s.n.], 2013. p. 142–153.

MANNING, C. D.; SURDEANU, M.; BAUER, J.; FINKEL, J.; BETHARD, S. J.; MCCLOSKY, D. The Stanford CoreNLP natural language processing toolkit. In: *Association for Computational Linguistics (ACL) System Demonstrations*. [s.n.], 2014. p. 55–60. Disponível em: <http://www.aclweb.org/anthology/P/P14/P14-5010>.

MANZATO, A. J.; SANTOS, A. B. *A elaboraÃ§Ã£o de questionÃ¡rios na pesquisa quantitativa*. [S.l.], 2012. DisponÃvel em http://www.inf.ufsc.br/.

MARCUS, M.; KIM, G.; MARCINKIEWICZ, M. A.; MACINTYRE, R.; BIES, A.; FERGUSON, M.; KATZ, K.; SCHASBERGER, B. The penn treebank: Annotating predicate argument structure. In: *Proceedings of the Workshop on Human Language Technology*. Stroudsburg, PA, USA: Association for Computational Linguistics, 1994. (HLT '94), p. 114–119. ISBN 1-55860-357-3. Disponível em: <https://doi.org/10.3115/1075812.1075835>.

MARTIN-RECUERDA, F.; WALTHER, D. Hys: Fast atomic decomposition and module extraction of owl-el ontologies. *Semantic Web Â– Interoperability, Usability, Applicability an IOS Press Journal*, 2015.

MILLER, G. A. Wordnet: A lexical database for english. *Communications of the ACM*, ACM, New York, NY, USA, v. 38, n. 11, p. 39–41, nov. 1995. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/219717.219748>.

MOORE, R. C. Semantical considerations on nonmonotonic logics. *Artificial Intelligence*, v. 25, p. 75–94, 1985.

MUSEN, M. A. The protégé project: a look back and a look forward. *AI Matters*, v. 1, n. 4, p. 4–12, 2015. Disponível em: <https://doi.org/10.1145/2757001.2757003>.

NEVES, J. M. A. C. Maria de F. Ontorevpro: Uma ontologia sobre revisÃ£o de programas para o aprendizado colaborativo de programaÃ§Ã£o em java. In: *Proceedings of the XVII SimpÃ³sio Brasileiro de InformÃ¡tica na EducaÃ§Ã£o*. [S.l.: s.n.], 2006. p. 569–578.

NORLIN, L.; FRANSSON, M. N.; ERIKSSON, M.; MERINO-MARTINEZ, R.; ANDERBERG, M.; KURTOVIC, S.; LITTON, J.-E. A minimum data set for sharing biobank samples, information, and data: Miabis. *Biopreservation and biobanking*, v. 10 4, p. 343–8, 2012.

NOY, N. F.; HAFNER, C. D. The state of the art in ontology design: A survey and comparative review. *AI Magazine*, v. 18, p. 53–74, 1997.

PARNAS, D. L. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, ACM, New York, NY, USA, v. 15, n. 12, p. 1053–1058, dez. 1972. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/361598.361623>.

PEREZ; CORCHO. Ontology Languages for the Semantic Web. *IEEE Intelligents Systems*, v. 13, p. 54–60, 2002.

PRESUTTI, V.; DAGA, E.; GANGEMI, A.; BLOMQVIST, E. extreme design with content ontology design patterns. In: *Proceedings of the 2009 International Conference on Ontology Patterns - Volume 516*. Aachen, Germany, Germany: CEUR-WS.org, 2009. (WOP'09), p. 83–97. Disponível em: <http://dl.acm.org/citation.cfm?id=2889761.2889768>.

OWLPizzas: Practical Experience of Teaching OWL-DL: Common Errors and Common Patterns. [S.l.]: Motta E., Shadbolt N.R., Stutt A., Gibbins N. (eds) Engineering Knowledge in the Age of the Semantic Web. EKAW 2004., 2004.

REN, Y.; PARVIZI, A.; MELLISH, C.; PAN, J. Z.; DEEMTER, K. van; STEVENS, R. Towards competency question-driven ontology authoring. In: PRESUTTI, V.; D'AMATO, C.; GANDON, F.; D'AQUIN, M.; STAAB, S.; TORDAI, A. (Ed.). *The Semantic Web: Trends and Challenges: 11th International Conference, ESWC 2014, Anissaras, Crete, Greece*. Cham: Springer International Publishing, 2014. p. 752–767. ISBN 978-3-319-07443-6. Disponível em: <http://dx.doi.org/10.1007/978-3-319-07443-6_50>.

RIBEIRO, M.; JUNIOR, P. M.; BORBA, P. Handbook of research on mobile software engineering: Design implementation and emergent applications. In: _____. [S.l.]: IGI Global, 2010. cap. Recommending Mechanisms for Modularizing Mobile Software Variabilities.

RODRIGUES, C.; FREITAS, F. Luiz GonÃ§alves de; BARREIROS, E. F. S.; AZEVEDO, R.; FILHO, A. Trigueiro de A. Legal ontologies over time: A systematic mapping study. *Expert Systems with Applications*, v. 130, 04 2019.

RUDOLPH, S. Foundations of Description Logics. In: *Proceedings of the 7th International Conference on Reasoning Web: Semantic Technologies for the Web of Data*. Berlin, Heidelberg: Springer-Verlag, 2011. (RW'11), p. 76–136. ISBN 978-3-642-23031-8. Disponível em: <http://dl.acm.org/citation.cfm?id=2033313.2033315>.

RUNESON, P. A survey of unit testing practices. *IEEE Software*, IEEE–Institute of Electrical and Electronics Engineers Inc., v. 23, n. 4, p. 22, 2006. ISSN 0740-7459.

SANTOS, P. M.; ROVER, A. J. Knowledge representation through ontologies: an application in the electronic democracy field. *Perspectivas em ciÃªncias da informaÃ§Ã£o*, scielo, v. 21, p. 22 – 49, 09 2016. ISSN 1413-9936. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1413-99362016000300022&nrm=iso>.

SIRIN, E.; PARSIA, B.; GRAU, B. C.; KALYANPUR, A.; KATZ, Y. Pellet: A practical owl-dl reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 5, n. 2, p. 51–53, jun. 2007. ISSN 1570-8268. Disponível em: <http://dx.doi.org/10.1016/j.websem.2007.03.004>.

SMITH, B.; WELTY, C. Fois introduction: Ontology-towards a new synthesis. In: *Proceedings of the International Conference on Formal Ontology in Information Systems - Volume 2001*. New York, NY, USA: ACM, 2001. (FOIS '01), p. 3–9. ISBN 1-58113-377-4. Conference Chair-Smith, Barry and Conference Chair-Welty, Christopher. Disponível em: <http://doi.acm.org/10.1145/505168.505201>.

STAAB, S.; STUDER, R.; SCHNURR, H.-P.; SURE, Y. Knowledge processes and ontologies. *IEEE Intelligent Systems*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 16, n. 1, p. 26–34, 2001. ISSN 1541-1672.

STUCKENSCHMIDT, H.; PARENT, C.; SPACCAPIETRA, S. *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2009. ISBN 3642019064, 9783642019067.

SUÁREZ-FIGUEROA, M. C.; GÓMEZ-PÉREZ, A.; FERNÁNDEZ-LÓPEZ, M. The neon methodology for ontology engineering. In: *Ontology Engineering in a Networked World*. [s.n.], 2012. p. 9–34. Disponível em: <http://dx.doi.org/10.1007/978-3-642-24794-1_2>.

SUÁREZ-FIGUEROA, M. C.; GÓMEZ-PÉREZ, A.; VILLAZÓN-TERRAZAS, B. How to write and use the ontology requirements specification document. In: *On the Move to Meaningful Internet Systems: OTM 2009, Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009, Vilamoura, Portugal,*

November 1-6, 2009, Proceedings, Part II. [s.n.], 2009. p. 966–982. Disponível em: <http://dx.doi.org/10.1007/978-3-642-05151-7_16>.

SUÁREZ-FIGUEROA, M. del C.; CEA, G. A. de; BUIL, C.; DELLSCHAFT, K.; FERNÁNDEZ-LÓPEZ, M.; GARCÍA, A.; GÓMEZ-PÉREZ, A.; HERRERO, G.; MONTIEL-PONSODA, E.; SABOU, M.; VILLAZON-TERRAZAS, B.; YUFEI, Z. Deliverable, *D5.4.1 NeOn Methodology for Building Contextualized Ontology Networks*. 2008.

TARJAN, R. E. Depth-first search and linear graph algorithms. *SIAM Journal on Computing (SICOMP)*, 1972.

USCHOLD, M.; GRUNINGER, M. Ontologies: principles, methods and applications. *The Knowledge Engineering Review*, v. 11, p. 93–136, 6 1996. ISSN 1469-8005. Disponível em: <http://journals.cambridge.org/article_S0269888900007797>.

USCHOLD, M.; KING, M. Towards a methodology for building ontologies. In: *In Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95*. [S.l.: s.n.], 1995.

VRANDEČIĆ, D.; GANGEMI, A. Unit tests for ontologies. In: MEERSMAN, R.; TARI, Z.; HERRERO, P. (Ed.). *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops: OTM Confederated International Workshops and Posters, Montpellier, France*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 1012–1020. ISBN 978-3-540-48276-5. Disponível em: <http://dx.doi.org/10.1007/11915072_2>.

W3C. *OWL Web Ontology Language Document Overview (Second Edition)*. 2012. Https://www.w3.org/TR/owl2-overview/. Ãºltimo acesso em 15 de janeiro de 2016.

ZIMMERMANN, A. Integrated distributed description logics. In: *Description Logics*. [S.l.: s.n.], 2007.

APÊNDICE A – OWL 2.0

Nesse apêndice constam os construtores e axiomas permitidos nas sublinguagens da OWL 2.0: OWL EL, OWL QL e OWL RL. Ele é baseado nas referências da W3C disponível em https://www.w3.org/TR/owl2-profiles/.

OWL EL Construtores:

- **ObjectSomeValuesFrom:** existencial quantification to a class expression
- ***DataSomeValuesFrom:*** existencial quantification to a data range
- **ObjectHasValue:** existencial quantification to an individual
- **DataHasValue:** existencial quantification to a literal
- **ObjectHasSelf:** self-restriction
- **ObjectOneOf:** enumerations involving a single individual
- **DataOneOf:** enumerations involving a single literal
- **ObjectIntersectionOf:** intersection of classes
- **DataIntersectionOf:** intersection of data ranges

Axiomas:

- **SubclassOf:** class inclusion
- **EquivalenceClasses:** class equivalence
- **DisjointClasses:** class disjointness
- **SubObjectPropertyOf:** object property inclusion with or without property chains
- **DataSubPropertyOf:** Data property inclusion
- **EquivalenceObjectProperties and EquivalenceDataProperties: property equivalence**
- **TransitiveObjectProperty:** transitive object properties
- **ReflexiveObjectProperty:** reflexive object properties
- **ObjectPropertyDomain and DataPropertyDomain:** domain restrictions
- **ObjectPropertyRange and DataPropertyRange:** range restrictions

- **SameIndividual, DifferentIndividuals, ClassAssertion, ObjectPropertyAssertion, DataPropertyAssertion, NegativeObjectPropertyAssertion and NegativeDataPropertyAssertion:** assertions
- **FunctionalDataProperty**: functional data properties
- **HasKey:** keys

OWL QL É uma linguagem baseada em DL-lite, que tem a intenção de prover raciocínio eficiente em grandes bancos de dados estruturados utilizando um esquema simples ((GRAU et al., 2008)). Possibilita a construção de uma conjunção de consultas que são respondidas em tempo *LogSpace*, usando um padrão de banco de dados relacional. Ela é útil em aplicações com um grande número de indivíduos e onde é necessário realizar consultas aos dados diretamente por meio de SQL ((W3C, 2012)).

OWL 2 QL restringe o local no qual certos construtores de expressões de classes podem ocorrer. Para expressões de subclasses:

- quantificador existencial(ObjectSomeValuesFrom);
- quantificador existencial para intervalo de dados (DataSomeValuesFrom).

Para expressões de superclasses:

- interseção(ObjectIntersectionOf);
- negação(ObjectComplementOf);
- quantificador existencial para uma classe(ObjectSomeValuesFrom);
- quantificador existencial para intervalo de dados(DataSomeValuesFrom).

Subclass Expressions	Superclass Expressions
a class existential quantification (**ObjectSomeValuesFrom**) where the class is limited to *owl:Thing* existential quantification to a data range (**DataSomeValuesFrom**)	a class intersection (**ObjectIntersectionOf**) negation (**ObjectComplementOf**) existential quantification to a class (**ObjectSomeValuesFrom**) existential quantification to a data range (**DataSomeValuesFrom**)

Figura 48 – Expressões de classes em OWL QL

Axiomas:

- **SubclassOf:** class inclusion
- **EquivalenceClasses:** class equivalence
- **DisjointClasses:** class disjointness

- **SubObjectPropertyOf:** object property inclusion with or without property chains
- **DataSubPropertyOf:** Data property inclusion
- **EquivalenceObjectProperties and EquivalenceDataProperties: property equivalence**
- **ReflexiveObjectProperty:** reflexive object properties
- **ObjectPropertyDomain and DataPropertyDomain:** domain restrictions
- **ObjectPropertyRange and DataPropertyRange:** range restrictions
- **DisjointObjectProperties and DisjointDataProperties:** disjoint properies
- **SymmetricObjectProperty:** symmetric properties
- **ReflexiveObjectProperty:** reflexive properties
- **Irreflexive properties:** irreflexive properties
- **AsymmetricObjectProperty:** asymmetric properties
- **DifferentIndividuals, ClassAssertion, ObjectPropertyAssertion, DataPropertyAssertion:** assertions

OWL RL

OWL 2 RL é direcionada para aplicações que precisam de um grande poder de raciocínio sem sacrificar muito poder de expressividade. Além disso, para aplicações RDFS que precisam da expressividade da OWL 2.

Esta linguagem possibilita a implementação de algoritmos de raciocínio em tempo polinominal por meio de regras estendidas de tecnologias de banco de dados operando diretamente sobre tuplas RDF.

Em OWL 2 RL, tarefas de raciocínio podem ser implementadas como um conjunto de regras na forma de um sistema que usa encadeamento para frente ((GRAU et al., 2008)).

OWL 2 RL permite a maioria dos construtores da OWL 2, porém, a forma como se usa tais construtores foi restringido. Por exemplo, restringem-se a axiomas de forma assimétrica, ou seja, podemos utilizar certos construtores como subclasses, mas não podemos utilizá-los como superclasses.

Axiomas:

Todos os axiomas da OWL 2.0 estão disponíveis, além de, disjoint unions of classes(DisjointUnion) e reflexive object property(ReflexiveObjectProperty)

Para maiores detalhes visitar o referencial da W3C em <https://www.w3.org/TR/2012/REC-owl2-profiles-20121211/>.

Subclass Expressions	**Superclass Expressions**
a class other than *owl:Thing* an enumeration of individuals (**ObjectOneOf**) intersection of class expressions (**ObjectIntersectionOf**) union of class expressions (**ObjectUnionOf**) existential quantification to a class expression (**ObjectSomeValuesFrom**) existential quantification to a data range (**DataSomeValuesFrom**) existential quantification to an individual (**ObjectHasValue**) existential quantification to a literal (**DataHasValue**)	a class other than *owl:Thing* intersection of classes (**ObjectIntersectionOf**) negation (**ObjectComplementOf**) universal quantification to a class expression (**ObjectAllValuesFrom**) existential quantification to an individual (**ObjectHasValue**) at-most 0/1 cardinality restriction to a class expression (**ObjectMaxCardinality** 0/1) universal quantification to a data range (**DataAllValuesFrom**) existential quantification to a literal (**DataHasValue**) at-most 0/1 cardinality restriction to a data range (**DataMaxCardinality** 0/1)

Figura 49 – Expressões de classes em OWL RL

APÊNDICE B – EXPERIMENTOS: QUESTÕES DE COMPETÊNCIA

Nesse apêndice consta as questões de competência utilizadas no questionário qualitativo.

Ontologia de Pizza

Which are the toppings of margherita pizza?

Resposta:

tomato, mozzarella, basil

subCQs:

What is pizza?

Pizza is Food

What is margheritapizza?

Margherita is a Pizza that has topping mozzarella , basil and tomato

Which are the toppings of ice cream?

Resposta: fruit

subCQs:

What is a icecream?

Icecream is a Food that has topping fruit

Icecream is disjoint of Margherita

Does American have mozzarella topping?

Resposta: yes

subCQs:

What is mozzarella?

Mozzarella is a Food

What is American?

American is Pizza

Is LaReine a pizza or a topping?

Resposta: Pizza
subCQs:
What is pizza?Pizza
is Food

What is LaReine?
LaReine is a Pizza

Ontologia de MadCow
Which food does a dog eat?**Resposta:**
bone
subCQs:
What is dog?
dog is animal that eats bone
domain of property “eats” is animal

What is food?
Apenas salvar

Who owns this cat?
Resposta:
catlover
subCQs:
What is cat?
cat is animal

Where is the heart?
Resposta:
chest

subCQs: What
is heart?

heart is an organ

What is sheep?

sheep is an animal that eats grass

What is brain?

brain is an organ that part of animal

range of property "part of" is animal

What is vegetarian?

vegetarian is an animal that does not eats only (animal and (part of animal))

What is cow?cow is vegetarian

What is madcow?

madcow is cow that eats brain and (part of sheep)

Ontologia de BreastCancer *What is a PersonUnderRisk?*

SubCqs:

What is PersonUnderRisk?

PersonUnderRisk is equivalent to a Person that hasRisk Risk

Is Alcohol a KnownFactor?

Answer: yes

SubCqs:

What is KnownFactor?

KnownFactor is RiskFactor

What is Alcohol?

Alcohol is KnownFactor

Which are the risk of BRCRType?

Answer: BreastCancer

SubCqs:

What is BRCRType?

BRCRType is equivalent to RiskType that riskOf BreastCancer

Does WomanWithBRCAMutation has risk?

Answer: yes

SubCqs:

What is WomanWithBRCAMutation?

WomanWithBRCAMutation is equivalent to Woman that hasRiskFactor BRCAMutation

What is risk?

Apenas salvar.

Is BRCRisk a risk?

Answer: Yes

SubCqs:

What is BRCRisk?

BRCRisk is Risk that riskType BRCRType

What is risk?

Apenas salvar

APÊNDICE C – *TAG PENN TREEBANK*

Abaixo o conjunto de tags disponíveis da NLP API, para encontrar a classe gramatical de uma palavra.

CC	Cordinating conjunction, e.g, and, but, or ...
CD	Cardinal number
DT	Determiner
EX	Existencial there
FW	Foreign word
IN	Preposion or subordinating conjuction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle

APÊNDICE D – WEB SEMÂNTICA

O uso de ontologias aumentou com o surgimento da *Web Semântica* proposta por (BERNERS-LEE; HENDLER; LASSILA, 2001), que tem como objetivo prover uma estrutura semântica às páginas Web de modo que não apenas os seres humanos, mas também agentes computacionais possam compreender os conteúdos contidos nelas. Um dos objetivos da Web Semântica é criar um ambiente onde agentes de software possam navegar pelos documentos Web realizando tarefas sofisticadas como, por exemplo, fornecer um material de aprendizagem adequado a um usuário que esteja fazendo avaliações em um sistema de aprendizagem de acordo com interações dele com o sistema e necessidades dele. A figura 50 mostra as camadas da Web Semântica, onde uma delas é a camada de ontologia. Ontologia é uma das tecnologias chave para tornar possível a *Web Semântica*, pois ela provê uma modelo semântico para especificação dos dados, possibilitando um vocabulário comum para troca de informação.

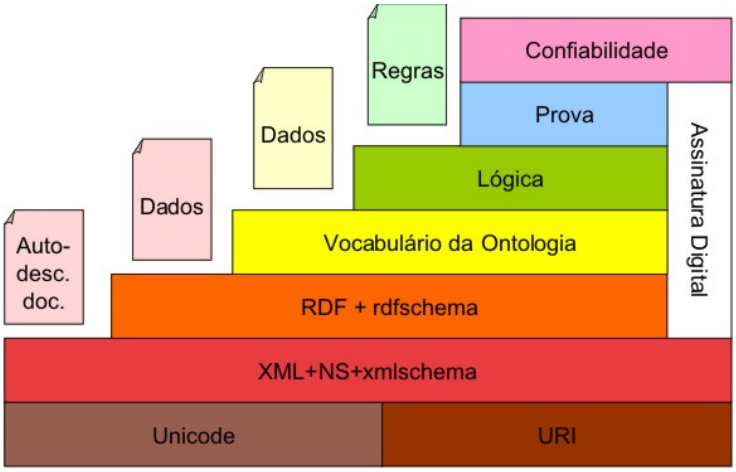


Figura 50 – Camadas da Web Semântica

Na base da arquitetura está o URI juntamente com o Unicode. O URI é um conjunto de caracteres para identificação de um recurso na Web. Um exemplo de URI é a URL que serve para referenciar um endereço na Web, como um site. O unicode é um padrão de codificação de caracteres que permite o intercâmbio de texto na *Web*.

Os *Namespaces* e a linguagem XML ficam acima da camada URI. Um *namespace* provê um método para evitar conflitos entre nomes de elementos da linguagem, atribuindo aos elementos nomes únicos. O nome de um *namespace* é um URI colocado geralmente no início do documento.

A XML consiste de uma linguagem de marcação de dados (metadados) que dá suporte a descrição de dados estruturados e de forma arbitrária, através de tags, facilitando declarações precisas do conteúdo de documentos. O principal construtor da linguagem é o *element* no qual se define um texto delimitado por tags ($<>$ *e* $</>$). Além disso, é possível associar atributos a elementos, que é composto por um nome e um valor. Porém, a linguagem XML apenas provê representação sintática de informações, não tendo suporte à estruturação semântica. Na figura 51 segue um exemplo de um código em xml, no qual foi definido informações de uma tese.

```
<?xml version = "1.0"?>
<tese>
 <title>Modularização de ontologias usando Questões de Competência</title>
 <author>
   <firstName>Camila</firstName>
   <lastName>Bezerra</lastName>
 </author>
 <advisor>
   <firstName>Fred</firstName>
   <lastName>Freitas</lastName>
 </coadvisor>
 <chapters>
    <chapter num = "1" pages = "4">Introdução</chapter>
    <chapter num = "2" pages = "30">Referencial teórico</chapter>
    <chapter num = "3" pages = "30">Proposta</chapter>
 </chapters>
</tese>
```

Figura 51 – Exemplo de XML

Em seguida vem a camada de RDF que também consiste em uma linguagem de representação, no entanto ela provê uma semântica simples para o que pretende representar, o que não pode ser realizado na linguagem XML. Ela foi desenvolvida essencialmente para representar metadados a respeito de recursos da Web.

O arquivo RDF contém uma estrutura principal de representação de recursos em RDF que é composta por três tipos de elementos: recursos,

propriedades e triplas. Um recurso é o que será descrito por uma expressão RDF. Todo recurso é identificado por um URI. Uma propriedade é uma característica utilizada para descrever um recurso. Uma tripla possui a forma < sujeito(recurso), predicado(propriedade), objeto(um valor que pode ser outro recurso) >. Por exemplo, o gatinho(sujeito) tem(propriedade) a cor amarela(valor). Outro exemplo, é exibido na figura 52 onde há dois recursos(www.cin.ufpe.br/ cbs, www.cin.ufpe.br/ fred) que são descritos, e conectados pela propriedade "orienta".

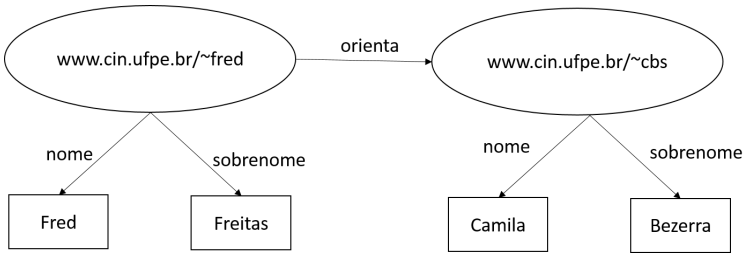


Figura 52 – Exemplo de RDF

RDF Schema é uma linguagem que define padrões(schemas) de vocabulário para definir propriedades e classes de recursos RDF, com uma semântica que permite, por exemplo, hierarquias de classes e propriedades. Na figura 53 um exemplo de uma representação gráfica do que podemos fazer com RDF Schema.

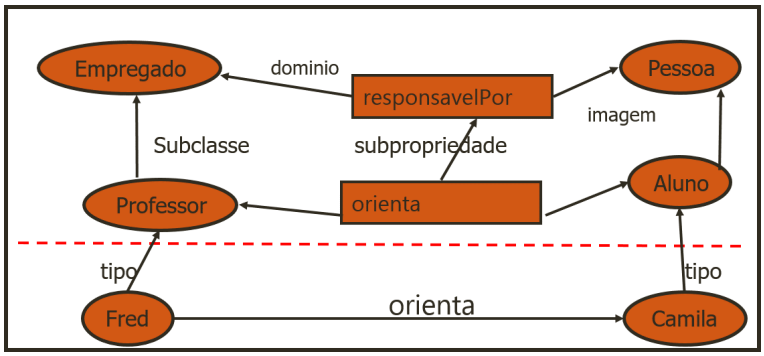


Figura 53 – Exemplo de RDF Shema

Uma das camadas da web semântica é a camada de ontologias que fornecem um conjunto de conceitos e termos para descrever um determinado domínio. As ontologias fornecem um vocabulário para a representação do conhecimento o qual possui uma conceitualização que o sustenta, evitando

interpretações ambíguas desse vocabulário ((NEVES, 2006)). Também é possível a reutilização de uma ontologia ou de parte de uma ontologia por outras aplicações que necessitem do mesmo domínio, dessa forma favorecendo o compartilhamento de conhecimento.

Como o RDF é uma linguagem que apresenta uma limitação de expressividade, W3C desenvolveu uma linguagem mais expressiva para Web Semântica, a OWL que é uma linguagem baseada em lógica de descrições, com o objetivo de descrever conhecimento rico e complexo geralmente acerca de um domínio em particular, que pode ser utilizada para criar ontologias.

A atual versão da OWL é 2.0 desenvolvida a partir de limitações signifi-cativas na expressividade da linguagem da versão anterior.

APÊNDICE E – LÓGICA DE DESCRIÇÕES

Na literatura existem várias linguagens de descrições, cada uma com um nível de expressividade diferente. Elas são diferenciadas pelos construtores que cada uma contém.

Linguagem $\mathcal{FL}^-$ Linguagem de descrição baseada em *frames* que permite:

- **Interseção de conceitos**: $C \sqcap D$, onde D e C são conceitos: Por exemplo,
  Interseção dos conceitos pessoa e aluno: Pessoa $\sqcap$ Aluno
- **Restrição universal**: $\forall R.C$, onde R é uma propriedade e C um conceito: Por exemplo,
  Pássaros que só tem a cor azul: $Passaro \sqcap \forall temCor.Azul$
- **Quantificador existencial limitado**: $\exists R.T$, onde R é uma propriedade e T é um conceito especial que enfatiza todas as instâncias.
  Conjunto de indivíduos que possuem pelo menos uma relação(conhece) em qualquer das instâncias da ontologia: $\exists conhece.T$

$\mathcal{FL}^-$ também apresenta algumas variações como a $\mathcal{FL}_{\mathcal{O}}$ que é uma sublinguagem de $\mathcal{FL}^-$, porém sem quantificador existencial limitado.

Linguagem $\mathcal{EL}$ A linguagem $\mathcal{EL}$ é um tipo de LD simples que contém os construtores exibidos na tabela 5.

Tabela 5 – Lógica $\mathcal{EL}$

Nome	Sintaxe	semântica
Conceito universal	$\top$	Δ^I
Interseção de conceitos	$C \sqcap D$	$C^I \cap D^I$
Quantificador existencial	$\exists R.C$	$\{d \in \Delta^I \mid \exists e \in C^I : (d, e) \in r^I\}$

$\mathcal{EL}$ não provê nenhum construtor e nem axioma de papel, cada papel em $\mathcal{EL}$ é atômico. Com relação a asserções, $\mathcal{EL}$ provê asserção de conceito (a:C) ou asserções de papel (r(a,b)). E os axiomas podem ser do tipo $C \sqsubseteq D$, $R \sqsubseteq S$, $C \equiv D$ ou $R \equiv S$.

Linguagem atributiva ($\mathcal{AL}$) A linguagem AL é a linguagem DL mais básica de interesse prático introduzida por Schmidt Schau and Gert Smolka em 1991 ((BAADER et al., 2010)). As outras linguagens são extensões de $\mathcal{AL}$.

Dado os conceitos A, C e D, e um papel R, descrições de conceitos são formadas pela sintaxe exibida na tabela 6:

Tabela 6 – Lógica $\mathcal{AL}$

Nome	Sintaxe	Semântica
Conceito universal	$\top$	Δ^I
Conceito inferior ou vazio	$\bot$	$\emptyset$
Negação atômica	$\neg$A	$\Delta^I \setminus C^I$
Intersecção de conceitos	$C \sqcap D$	$C^I \cap D^I$
Restrição universal	$\forall R.C$	$\neg(\exists r.\neg C)^I$
$\exists R.T$	quantificador existencial limitado	$\exists$ R(x,y)

Linguagem $\mathcal{ALC}$ Um outro tipo de linguagem DL é ALC também denominada por $\mathcal{S}$ com a adição de papéis transitivos. A linguagem $\mathcal{ALC}$ é composta pelos construtores: negação ($\neg$), conjunção($\sqcap$), disjunção ($\sqcup$), quantificador existencial ($\mathcal{E}$) e quantificador universal ($\forall$), como mostra a tabela 7.

Tabela 7 – Lógica $\mathcal{ALC}$

Sintaxe	Nome	semântica
$\top$	conceito universal	Δ^I
$\bot$	conceito inferior ou vazio	$\emptyset$
$\neg$A	negação atômica	$\Delta^I \setminus C^I$
$C \sqcap D$	conjunção de conceitos	$C^I \sqcap D^I$
$C \sqcup D$	união de conceitos	$\neg(\neg C^I \sqcup \neg D^I)^I$
$\forall R.C$	restrição universal	$\neg(\exists r.\neg C)^I$
$\exists R.C$	restrição existencial	$\{d \in \Delta^I \mid \exists e \in C^I : (d,e) \in r^I\}$

Linguagem $\mathcal{ALCN}$ Um outro tipo de linguagem LD é ALCN. $\mathcal{ALCN}$ é composta pelos construtores: negação ($\neg$), conjunção($\sqcap$), disjunção ($\sqcup$), quantificador existencial ($\mathcal{E}$), quantificador universal ($\forall$), e restrições de número como mostra a tabela 8.

Tabela 8 – Sintaxe x Semântica de $\mathcal{ALCN}$.

Construtor	**Sintaxe**	**Semântica**
Indivíduos		
Nome do Indivíduo	a	a^I
Papéis		
Relação	R	R^I
Conceitos		
Conceito	A	A^I
Intersecção	$\mathsf{C} \sqcap \mathsf{D}$	$\mathsf{C}^I \cap \mathsf{D}^I$
União	$\mathsf{C} \sqcup \mathsf{D}$	$\mathsf{C}^I \cup \mathsf{D}^I$
Complemento	$\neg\mathsf{C}$	$\Delta^I \setminus \mathsf{C}^I$
Top Concept	$\top$	Δ^I
Bottom Concept	$\bot$	$\{\ \}$
Restrição Existencial	$\exists\mathsf{R}.\mathsf{C}$	$\{\mathsf{x} \in \Delta^I \mid \exists\ \mathsf{y}, (\mathsf{x},\mathsf{y}) \in \mathsf{R}^I \text{ and } \mathsf{y} \in \mathsf{C}^I\}$
Restrição Universal	$\forall\mathsf{R}.\mathsf{C}$	$\{\mathsf{x} \in \Delta^I \mid \forall\ \mathsf{y}, (\mathsf{x},\mathsf{y}) \in \mathsf{R}^I \Rightarrow \mathsf{y} \in \mathsf{C}^I\}$
Restrição Numérica	$\geq n\mathsf{R}.\mathsf{C}$	$\{\mathsf{x} \in \Delta^I \mid \exists$ ao menos $\mathsf{n}\ \mathsf{y}, (\mathsf{x},\mathsf{y}) \in \mathsf{R}^I \text{ and } \mathsf{y} \in \mathsf{C}^I\}$
	$\leq n\mathsf{R}.\mathsf{C}$	$\{\mathsf{x} \in \Delta^I \mid \exists$ no máximo $\mathsf{n}\ \mathsf{y}, (\mathsf{x},\mathsf{y}) \in \mathsf{R}^I \text{ and } \mathsf{y} \in \mathsf{C}^I\}$