

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/338720095>

Guia prático de construção de ontologias – Protégé v. 5.2

Method · March 2018

DOI: 10.13140/RG.2.2.31922.96969

CITATIONS

0

READS

278

1 author:



Jean-Claude Miroir

University of Brasília

14 PUBLICATIONS 4 CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:



Translation teaching methods [View project](#)



Criticism of translation [View project](#)

GUIA PRÁTICO DE CONSTRUÇÃO DE ONTOLOGIAS OWL

ADAPTADO PARA PROTÉGÉ 5.2

Jean-Claude Miroir

Universidade de Brasília (UnB)

Março 2018

Matthew Horridge

Contributors

v1.0 Holger Knublauch , Alan Rector , Robert Stevens , Chris Wroe

v1.1 Simon Jupp, Georgina Moulton, Robert Stevens

v1.2 Nick Drummond, Simon Jupp, Georgina Moulton, Robert Stevens

v1.3 Sebastian Brandt

The University Of Manchester

Copyright © The University Of Manchester

March 24, 2011

Original traduzido e adaptado:

HORRIDGE, M. et al. **A Practical Guide To Building OWL Ontologies using the Protege-OWL plugin and CO-ODE Tools, Edition 1.0.** (2004). Available: <<http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf>>. Access: 20 June 2005.

Acesso atualizado: Disponível em: <http://mowl-power.cs.man.ac.uk/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf>. Acesso em: 11 fev. 2018.

Referência da tradução da versão de Protégé-OWL 3.4:

HORRIDGE, M. et al. **Um guia prático para a construção de ontologias OWL, plugin Protégé-OWL 3.4.** Tradução de SOARES, D.R.; ALMEIDA, M.B. Disponível em: <www.eci.ufmg.br/mba/>, 2008. 100p. (do original em inglês).

Acesso atualizado: Disponível em: <<http://mba.eci.ufmg.br/onto.owl/>>. Acesso em: 11 fev. 2018.

Lista dos exercícios

Exercício 1: Instalação de Protégé 5	12
Exercício 2: Criar uma nova ontologia OWL no Protégé 5	16
Exercício 3: Adicionar um comentário geral sobre a ontologia criada	18
Exercício 4: Criar 3 classes: Pizza, RecheioDePizza (PizzaTopping), MassaDePizza (PizzaBase).....	20
Exercício 5: Tornar disjuntas as classes Pizza, RecheioDePizza (PizzaTopping), MassaDePizza (PizzaBase)	23
Exercício 6: Usar o assistente para criar as classes MassaFinaECrocante (ThinAndCrispyBase) e MassaGrossa (DeepPanBase) como subclasses de MassaDePizza (PizzaBase).....	24
Exercício 7: Criar algumas subclasses de RecheioDePizza (PizzaTopping).....	26
Exercício 8: Criar uma propriedade de objeto chamada hasIngredient	31
Exercício 9: Criar duas subpropriedades de temIngrediente (hasIngredient): temRecheio (hasTopping) e temMassa (hasBase).....	32
Exercício 10: Criar as propriedades inversas de temIngredient, de temMassa, de temRecheio	34
Exercício 11: Tornar transitiva a propriedade temIngrediente (hasIngredient)	40
Exercício 12: Tornar funcional a propriedade temMassa (hasBase).....	40
Exercício 13: Especificar a classe RecheioDePizza (PizzaTopping) como o escopo (Range) da propriedade temRecheio (hasTopping)	43
Exercício 14: Especificar Pizza o domínio (<i>Domain</i>) da propriedade temRecheio (hasTopping)	44
Exercício 15: Especificar o domínio (Domain) e o escopo (Range) da propriedade temMassa (hasBase) e de sua propriedade inversa éMassaDe (isBaseOf).....	47
Exercício 16: Adicionar uma restrição à classe Pizza que especifica que a classe Pizza deve ter uma MassaDePizza (PizzaBase) [parte 1]	51
Exercício 17: Adicionar uma restrição à classe Pizza que especifica que a classe Pizza deve ter uma MassaDePizza (PizzaBase) [Cont. - Parte2]	52
Exercício 18: Criar uma subclasse de Pizza chamada PizzaComNome (NamedPizza), e uma subclasse de PizzaComNome (NamedPizza) chamada PizzaMarguerita (MargheritaPizza)	54
Exercício 19: Criar uma restrição existencial (some) para a classe PizzaMarguerita (MargheritaPizza) que age por meio da propriedade temRecheio (hasTopping) com o complemento da restrição (<i>filler</i>) de RecheioDeMuçarela (MozzarellaTopping) para especificar que uma PizzaMarguerita (MargheritaPizza) possui pelo menos um RecheioDeMuçarela (MozzarellaTopping)	56
Exercício 20: Criar uma restrição existencial (some) para a classe PizzaMarguerita (MargheritaPizza) que age por meio da propriedade temRecheio (hasTopping) com o complemento da restrição (<i>Filler</i>) RecheioDeTomate (TomatoTopping) para especificar que uma PizzaMarguerita (MargheritaPizza) possui pelo menos um RecheioDeTomate (TomatoTopping).....	57
Exercício 21: Criar a classe PizzaAmericana (AmericanPizza), clonando a classe PizzaMarguerita (MargheritaPizza) e modificando sua descrição, acrescentando um recheio: RecheioDeCalabresa (PepperoniTopping).....	58

Exercício 22: Criar as classes PizzaAmericanaPicante (AmericanHotPizza) e PizzaSoho (SohoPizza), por clonagem	59
Exercício 23: Tornar disjuntas todas as subclasses da classe PizzaComNome (NamedPizza)	61
Exercício 24: Adicionar uma classe de investigação (Probe Class) denominada InvestigaçāoInconsistenteDoRecheio (ProbeInconsistentTopping) como subclasse de RecheioDeQueijo (CheeseTopping) e de RecheioDeVerduras (VegetableTopping).....	63
Exercício 25: Verificar a inconsistência da classe InvestigaçāoInconsistenteDoRecheio (ProbeInconsistentTopping)	64
Exercício 26: Remover a declaração disjunta entre as classes RecheioDeQueijo (CheeseTopping) e RecheioDeVerduras (VegetableTopping) para observar o resultado..	66
Exercício 27: Corrigir a ontologia, tornando disjuntas as classes RecheioDeQueijo (CheeseTopping) e RecheioDeVerduras (VegetableTopping).....	67
Exercício 28: Criar uma subclasse de Pizza chamada PizzaDeQueijo (CheesyPizza), com pelo menos um recheio que é um tipo de RecheioDeQueijo (CheeseTopping)	69
Exercício 29: Converter as condições necessárias da classe PizzaDeQueijo (CheesyPizza) em condições necessárias e suficientes	70
Exercício 30: Usar o mecanismo de inferência (<i>Reasoner</i>) para computar automaticamente a subclasse de PizzaDeQueijo (CheesyPizza)	74
Exercício 31: Criar uma classe para descrever uma PizzaVegetariana (VegetarianPizza)	77
Exercício 32: Converter as condições necessárias da classe PizzaVegetariana (VegetarianPizza) em condições necessárias e suficientes	79
Exercício 33: Usar o mecanismo de inferência (<i>Reasoner</i>) para computar automaticamente a classe PizzaVegetariana (VegetarianPizza)	80
Exercício 34: Adicionar um axioma de fechamento (Closure axiom) à propriedade temRecheio (hasTopping) para PizzaMarguerita (MargheritaPizza)	81
Exercício 35: Adicionar um axioma de fechamento (Closure axiom) à propriedade temRecheio (hasTopping) para PizzaSoho (SohoPizza)	83
Exercício 36: Criar automaticamente um axioma de fechamento na propriedade temRecheio (hasTopping) para a classe PizzaAmericana (AmericanPizza)	84
Exercício 37: Criar automaticamente um axioma de fechamento na propriedade temRecheio (hasTopping) para a classe PizzaAmericanaPicante (AmericanHotPizza)	84
Exercício 38: Usar o mecanismo de inferência (<i>Reasoner</i>) para computar as subclasses de PizzaDeQueijo e de PizzaVegetariana	85
Exercício 39: Criar uma partição de valor (Value Partition) para representar o picante (spiciness) em recheios de Pizza.....	87
Exercício 40: Especificar a propriedade restritiva temPicante (hasSpiciness) nas classes de recheios de pizzas	92
Exercício 41: Criar uma classe PizzaPicante (SpicyPizza) como uma subclasse de Pizza .	94
Exercício 42: Usar o mecanismo de inferência (<i>Reasoner</i>) para classificar a ontologia ...	96
Exercício 43: Criar uma classe PizzaInteressante (InterestingPizza) com pelo menos 3 recheios	97
Exercício 44: Usar o mecanismo de inferência (<i>Reasoner</i>) para classificar a ontologia ...	98
Exercício 45: Criar uma classe PizzaQuatroQueijos (FourCheesePizza) que tenha exatamente quatro recheios (diferentes) de queijo	99
Exercício 46: Criar uma propriedade do tipo de dados (<i>Data Type Properties</i>) denominada temValorDoTeorCalórico (hasCalorificContentValue).....	101
Exercício 47: Criar exemplos de indivíduos de pizzas	102

Exercício 48: Criar uma restrição de tipo de dados (<i>DataType</i>) para indicar que todas as pizzas têm um valor energético (expresso em calorias)	105
Exercício 49: Criar uma classe PizzaDeTeorCalóricoElevado (HighCaloriePizza) que tenha um valor energético superior ou igual a 400 calorias	106
Exercício 50: Classificar os indivíduos de pizzas de acordo com os critérios da propriedade temValorDoTeorCalórico (hasCalorificContentValue).....	108
Exercício 51: Tornar funcional a propriedade do tipo de dados (<i>DataType Properties</i>) temValorDoTeorCalórico (hasCalorificContentValue).....	110
Exercício 52: Criar uma classe PizzaNãoVegetariana (NonVegetarianPizza), subclasse da classe Pizza e disjunta da classe PizzaVegetariana (VegetarianPizza)	112
Exercício 53: Tornar complemento da classe PizzaVegetariana (VegetarianPizza) a classe PizzaNãoVegetariana (NonVegetarianPizza)	113
Exercício 54: Adicionar a classe Pizza às condições necessárias e suficientes da classe PizzaNãoVegetariana (NonVegetarianPizza)	114
Exercício 55: Usar o mecanismo de inferência (<i>Reasoner</i>) para computar as classes PizzaNãoVegetariana e PizzaVegetariana	115
Exercício 56: Criar uma classe PizzaSemFechamento (UnclosedPizza), subclasse de PizzaComNome (NamedPizza), com um recheio de Muçarela (Mozzarella).....	115
Exercício 57: Usar o mecanismo de inferência (<i>Reasoner</i>) para classificar a ontologia .	116
Exercício 58: Criar uma classe chamada País (Country) e povoar esta classe de alguns indivíduos.....	118
Exercício 59: Criar uma restrição temValor (hasValue) para especificar que RecheioDeMuçarela (MozzarellaTopping) tem Itália (Italy) como país de origem.....	120
Exercício 60: Converter a classe País (Country) em uma classe enumerada (<i>Enumerated classes</i>).....	122

Sumário

1. Introdução.....	11
1.1. Convenções	11
2. Requisitos.....	12
3. O que são as ontologias OWL?.....	12
3.1. As três espécies de OWL.....	12
3.1.1. OWL-Lite.....	13
3.1.2. OWL-DL.....	13
3.1.3. OWL-Full.....	13
3.1.4. A escolha da sublinguagem	13
3.2. Componentes da Ontologia OWL.....	13
3.2.1. Indivíduos (<i>Individuals</i>)	14
3.2.2. Propriedades (<i>Properties</i>).....	14
3.2.3. Classes (Classes)	15
4. Construção de uma ontologia OWL	16
4.1. Criação de classes nomeadas (Named classes)	19
4.2. Disjunção de classes.....	22
4.3. Criação de classes com o assistente “Criar Hierarquia de Classe...”	24
4.3.1. Criação de algumas subclasses de RecheioDePizza (PizzaTopping)	26
4.4. Propriedades em OWL	30
4.5. Propriedades inversas (Inverse properties)	34
4.6. Características das propriedades de objetos (Object Properties)	37
4.6.1. Propriedades funcionais (<i>Functional properties</i>)	37
4.6.2. Propriedades funcionais inversas (<i>Inverse Functional properties</i>)	38
4.6.3. Propriedades transitivas (<i>Transitive Properties</i>).....	38
4.6.4. Propriedades simétricas (<i>Symmetric properties</i>)	39
4.6.5. Propriedades assimétricas (<i>Asymmetric properties</i>)	41
4.6.6. Propriedades reflexivas (<i>Reflexive properties</i>).....	41
4.6.7. Propriedades irreflexivas (<i>Irreflexive properties</i>)	42
4.7. Domínios (Domains) e Escopos (Ranges) de uma propriedade.....	42
4.8. Descrição e definição de classes.....	48
4.8.1. Restrições de propriedades.....	48
4.8.2. Restrições existenciais	51
4.9. Uso de um mecanismo de inferência (Reasoner)	62
4.9.1. Conhecendo o mecanismo de inferência (MI)	62
4.9.2. Classes inconsistentes	63

4.10. Condições necessárias e suficientes (classes primitivas e definidas) ...	68
4.11. Classificação automática	74
4.12. Restrições universais.....	76
4.13. Classificação automática e Raciocínio de Mundo Aberto (Open World Reasoning – OWR)	79
4.13.1. Axiomas de fechamento (<i>Closure axiom</i>).....	81
4.14. Partições de valores (Value Partitions)	86
4.14.1. Axiomas de cobertura (Covering axioms).....	90
4.15. Acrescentar picante aos recheios de pizzas	92
4.16. Restrições de cardinalidade (Cardinality Restrictions)	96
4.17. Restrições de cardinalidade qualificadas (Qualified Cardinality Restrictions).....	98
5. Propriedades de tipo de dados (DataType Properties).....	100
6. Mais sobre o raciocínio de mundo aberto (Open World Reasoning – OWR)	112
7. Criar outras estruturas OWL no Protégé	118
7.1. Criar indivíduos (<i>Individuals</i>)	118
7.2. Restrição temValor (<i>hasValue</i>).....	120
7.3. Classes enumeradas (<i>Enumerated classes</i>)	121
7.4. Propriedades de Anotação (<i>Annotation Properties</i>).....	123
7.5. Conjuntos múltiplos de Condições Necessárias e Suficientes	125
8. Apêndice A: Tipos de restrições (Restriction types)	127
8.1. Restrições de quantificação (Quantifier Restrictions).....	127
8.1.1. Restrições Existenciais (<i>Existential Restrictions</i>): someValuesFrom	127
8.1.2. Restrições Universais (<i>Universal Restrictions</i>): allValuesFrom	129
8.1.3. Combinação de restrições Existenciais e Universais na descrição de classes	129
8.2. Restrições hasValue (hasValue Restrictions).....	130
8.3. Restrições de cardinalidade (Cardinality Restrictions)	131
8.3.1. Restrições de cardinalidade mínima	131
8.3.2. Restrições de cardinalidade máxima	131
8.3.3. Restrições de cardinalidade exata.....	131
8.3.4. Postulado de nome único (<i>Unique Name Assumption — UNA</i>) e cardinalidades	132
9. Apêndice B: Descrições de classes complexas (Complex Class Descriptions)	132
9.1.1. Classes de interseção (\cap).....	132
9.2. Classes de união (\cup).....	133
10. Referências bibliográficas.....	134

Informações liminares

Original traduzido e adaptado:

HORRIDGE, M. et al. **A Practical Guide To Building OWL Ontologies using the Protege-OWL plugin and CO-ODE Tools, Edition 1.0.** (2004). Available from Internet: <<http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf>>. Access: 20 June 2005.

Acesso atualizado: Disponível em: <http://mowl-power.cs.man.ac.uk/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf>. Acesso: 11 fev. 2018.

Referência da tradução:

HORRIDGE, M. et al. *Um guia prático para a construção de ontologias OWL, plugin Protégé-OWL 3.4.* Trad. SOARES, D.R.; ALMEIDA, M.B. Disponível na Internet <www.eci.ufmg.br/mba/>, 2008. 100p. (Original inglês).

Acesso atualizado: Disponível em: <<http://mba.eci.ufmg.br/onto.owl/>>. Acesso: 11 fev. 2018.

Conforme o aviso original dos tradutores (Soares e Almeida), para o português do Brasil, da versão 1.0 desse guia (HORRIDGE, 2008) (original em inglês), que consta no quadro acima, a versão da ferramenta usada é o *plugin Protégé-OWL 3.4*. Ele foi redigido, em 2004, por Matthew Horridge e colaboradores (HORRIDGE et al., 2004) (da Universidade de Stanford e da Universidade de Manchester). A ontologia de Pizza, desenvolvida neste guia, tornou-se uma aplicação didática de renome mundial.

A versão atual (fevereiro 2018) de Protégé, disponível no site oficial da ferramenta, desenvolvida pela Universidade de Stanford, corresponde à versão 5.2.

Protégé 5 sofreu mudanças significativas, não apenas do ponto de vista ergonômico, mas também, no que se refere a suas funcionalidades, que se adaptaram, de certa forma, à evolução da própria linguagem OWL (*Ontology Web Language*). Tornou-se, portanto, mais difícil realizar os exercícios propostos, seguindo o tutorial para a versão 3.4 de Protégé (em inglês ou em português), para realizá-los com a versão atual de Protégé 5.

O presente guia foi adaptado, por mim, para a nova realidade, baseando-se nas três versões do tutorial original em inglês (2004, 2009, 2011) e na tradução de 2008. De fato, as versões de 2009 e de 2011 foram atualizadas para a versão Protégé 4, mais parecida com Protégé 5 do que a versão 3.4.

Todos os exercícios, que constam neste guia, foram realizados com a versão 5.2 do Protégé, conforme as orientações disponíveis no guia em inglês de 2011 (HORRIDGE; BRANDT, 2011), usando a maior parte da tradução para o português do Brasil de 2008 e adaptando-a ou retraduzindo o original em inglês (2011) quando a nova realidade o exigia. Por exemplo, o capítulo 5 “Propriedades de tipo de dados (*Data Type Properties*)” não existia na primeira versão de 2004 e foi traduzido por mim. Algumas partes que foram excluídas da versão de 2011, em relação à primeira versão, foram reintegradas conforme sua pertinência com a versão atual de Protégé 5. Essas “reinserções” são destacadas em notas de rodapé.

Todas as classes, propriedades, anotações, entre outras, foram traduzidas para o português do Brasil, a fim de que a ontologia possa ser explorada neste idioma.

Espero que esse guia possa cumprir, em português do Brasil (baseado na tradução de Soares e Almeida de 2008), os objetivos didáticos definidos por seus idealizadores, para explorar a versão 5.2 de Protégé.

Jean-Claude Miroir

Brasília, março de 2018.

1. Introdução

O presente guia descreve a criação de ontologias utilizando o editor de ontologia *Protégé 5* (versão 5.2.0).

Apresenta-se brevemente a linguagem OWL (*Ontology Web Language*), uma linguagem baseada em Lógica Descritiva, e enfatiza-se a construção de ontologias OWL-DL.

Utiliza-se um Mecanismo de Inferência (*Reasoner*) baseado em lógica descritiva para verificar a consistência da ontologia e para computar automaticamente hierarquia de classes.

Além disso, descrevem-se conceitos OWL, tais como a restrição **temValor** e as Classes enumeradas (*Enumerated classes*), descrevem-se as características e funcionalidades da ferramenta *Protégé 5*.

1.1. Convenções

Os exercícios são apresentados da seguinte forma:

Exercício 0: Faça o seguinte:

- 1º. Faça isso.
- 2º. Em seguida, faça isso.
- 3º. Em seguida, faça isso.

Outras convenções utilizadas:

- ◆ Elementos da ontologia (Classes, Propriedades, entre outras), traduzidos em português do Brasil (**com formatação específica**) e o original em inglês. Por exemplo: **RecheioDePizza** (*PizzaTopping*).
- ◆ Funções do Protégé 5 (comandos, ícones, entre outros), em inglês (**com formatação específica**) e traduzidos em português do Brasil. Por exemplo: **Active Ontology** (Ontologia ativa).
- ◆ Seleção de um conjunto de comandos, ícones, entre outros: **Enviar para > Área de trabalho (criar atalho)**
- ◆ Nome de arquivo (com sua extensão). Por exemplo: **Protege-5.2.0-win.zip**
- ◆ Nome de pasta. Por exemplo: **Download**
- ◆ Elementos de código, caminho formal: Por exemplo: `c:\Arquivos de Programas (x86)`

 Dicas.

 Observações importantes.

Terminologia específica.

2. Requisitos

<http://Protégéproject.github.io/Protégé/installation/windows/>

Exercício 1: Instalação de Protégé 5

- 1º. Acessar à página oficial do Protégé para baixar o programa, **Protege-5.2.0-win.zip**: <https://protege.stanford.edu/products.php#desktop-protege>
- 2º. Após localizar o arquivo (em geral na pasta **Download**), extrair o arquivo ZIP, por exemplo, na pasta c:\Arquivos de Programas (x86)
- 3º. Abrir a pasta **Protege-5.2.0-win**. Para lançar Protégé 5, clicar (duas vezes) no arquivo **Protege.exe**
- 4º. É possível criar um atalho para a **Área de trabalho**, clicando com o botão direito do mouse no arquivo **Protege.exe**, selecionar **Enviar para > Área de trabalho (criar atalho)**

3. O que são as ontologias OWL?

As ontologias são utilizadas para capturar conhecimento sobre um domínio de interesse. Uma ontologia descreve os conceitos de um domínio e também as relações que existem entre esses conceitos. Há diversas linguagens para construção de ontologias que fornecem diferentes funcionalidades. O padrão mais recente de linguagens para ontologias é o OWL, desenvolvido no âmbito do *World Wide Web Consortium* (W3C).

O Protégé 5 possui um amplo conjunto de operadores, por exemplo, a interseção (*OR*), união (*AND*) e negação (*NOT*) e é baseado em um modelo lógico que torna possível definir conceitos da forma como são descritos. Conceitos complexos podem ser constituídos a partir de definições de conceitos simples.

Além disso, o modelo lógico permite a utilização de um mecanismo de inferência (MI), o qual pode verificar se as declarações e as definições da ontologia são mutuamente consistentes entre si e reconhecer se conceitos são adequados a definições. O MI pode, portanto, ajudar a manter a hierarquia, o que é útil quando existem casos em que uma classe tem mais de um “pai”.

3.1. As três espécies de OWL

As ontologias OWL podem ser classificadas em três espécies, de acordo com a sublinguagem utilizada: *OWL-Lite*, *OWL-DL* e *OWL-Full*. A característica principal de cada sublinguagem é a sua expressividade: a *OWL-Lite* é a menos expressiva; a *OWL-Full* é a mais expressiva; a expressividade da *OWL-DL* está entre as duas, entre a *OWL-Lite* e a *OWL-Full*.

3.1.1. OWL-Lite

A *OWL-Lite* é a sublinguagem sintaticamente mais simples. Destina-se a situações em que apenas são necessárias restrições e uma hierarquia de classe simples. Por exemplo, o *OWL-Lite* pode fornecer uma forma de migração para tesouros existentes, bem como para outras hierarquias simples.

3.1.2. OWL-DL

A *OWL-DL* é mais expressiva que a *OWL-Lite* e baseia-se em lógica descritiva, um fragmento de Lógica de Primeira Ordem, passível, portanto, de raciocínio automático. É possível assim computar automaticamente a hierarquia de classes e verificar inconsistências na ontologia. Este tutorial utiliza a *OWL-DL*.

3.1.3. OWL-Full

A *OWL-Full* é a sublinguagem OWL mais expressiva. Destina-se a situações onde a alta expressividade é mais importante do que garantir a decidibilidade ou completeza da linguagem. Não é possível efetuar inferências em ontologias *OWL-Full*.

3.1.4. A escolha da sublinguagem

Para maiores detalhes sobre as três sublinguagens OWL, ver informações no W3C. Embora muitos fatores interfiram na escolha da sublinguagem adequada, existem algumas regras básicas:

- ➔ Entre *OWL-Lite* e *OWL-DL*, é necessário saber se os conceitos da *OWL-Lite* são suficientes;
- ➔ Entre *OWL-DL* e *OWL-Full*, é preciso saber se é importante realizar inferências na ontologia, ou se é importante usar funcionalidades altamente expressivas ou funcionalidades de modelagem, tais como as meta-classes (classes de classes).

3.2. Componentes da Ontologia OWL

As ontologias OWL têm componentes similares à estrutura do *Frame-based Protégé*, mas a terminologia usada para descrever tais componentes é um pouco diferente da utilizada no *Protégé-Frames*. A correspondência entre as duas nomenclaturas é apresentada no quadro seguinte:

Protégé - Frames	Protégé - OWL
Instâncias (<i>Instances</i>)	Indivíduos (<i>Individuals</i>)
Slots (<i>Slots</i>)	Propriedades (<i>Properties</i>)
Classes (<i>Classes</i>)	Classes (<i>Classes</i>)

Quadro 3.1 - Correspondência entre nomenclaturas

3.2.1. Indivíduos (*Individuals*)

Indivíduos representam objetos no domínio de interesse (ou domínio do discurso). Uma diferença importante entre o *Protégé* e o *OWL* é que este último não usa o *Unique Name Assumption* (UNA). Isto significa que dois nomes diferentes podem remeter ao mesmo indivíduo. Por exemplo, *Queen Elizabeth* (Rainha Elizabeth), *The Queen* (A rainha) e, *Elizabeth Windsor* podem se referir ao mesmo indivíduo. Em *OWL* deve-se declarar explicitamente que os indivíduos são os mesmos, ou diferentes uns dos outros. A Figura 3.1 (abaixo) mostra uma representação de alguns indivíduos em alguns domínios. Neste tutorial representam-se os indivíduos na forma de losangos.



Figura 3.1 – Representação dos indivíduos

Os *indivíduos* são também conhecidos como *instâncias*. Os indivíduos podem ser referenciados como *Instâncias de classes*.

3.2.2. Propriedades (*Properties*)

Propriedades¹ são relações binárias² entre indivíduos, ou seja, as propriedades ligam dois indivíduos³. Por exemplo, a propriedade **hasSibling** (temIrmão) pode ligar o indivíduo **Matthew** ao indivíduo **Gemma**; ou a propriedade **hasChild** (temCriança) pode ligar o indivíduo **Peter** ao indivíduo **Matthew**. As Propriedades podem também ser inversas. Por exemplo, a propriedade inversa de **hasOwner** (temDono) é **isOwnedBy** (éPropriedadeDe).

As propriedades podem limitar-se a um valor único: são as propriedades funcionais (*Functional properties*). Elas também podem ser propriedades transitivas (*Transitive Properties*) ou propriedades simétricas (*Symmetric Properties*). Estas características das propriedades são detalhadas adiante. A Figura 3.2 mostra propriedades que conectam indivíduos.

¹ Propriedades são equivalentes aos slots no Protege-Frames. Também são conhecidas como papéis (roles) em lógica descritiva, como relações (relationships) em UML-Unified Modeling Language e em outras abordagens de orientação a objeto. Em muitos formalismos, como no GRAIL, elas são denominadas de atributos.

² Uma relação binária é uma relação entre duas coisas

³ Em termos estritos, deve-se falar de "instâncias de propriedades" que ligam os indivíduos, mas, por uma questão de brevidade, será simplificado.

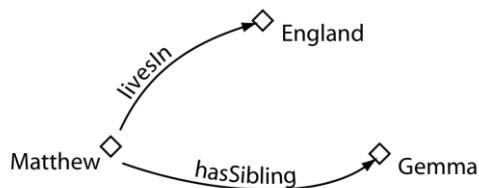


Figura 3.2 – Representação de propriedades

3.2.3. Classes (Classes)

As classes OWL são conjuntos que contêm os indivíduos. Elas são descritas formalmente (descrições matemáticas) a fim de que sejam apresentados os requisitos para a participação na classe. Por exemplo, a classe **Cat** (Gato) pode conter todos os indivíduos que são gatos, no domínio de interesse.

As classes podem ser organizadas em hierarquias superclasse-subclasse, também conhecidas como taxonomias. Subclasses são especializações de suas superclasses. Por exemplo, considere-se as classes **Animal** e **Cat**: **Cat** pode ser subclass de **Animal**, e assim **Animal** é superclasse de **Cat**.

Isso quer dizer que:

- Todos os Gatos são Animais;
- Todos os membros da classe **Cat** (Gato) são membros da classe **Animal**;
- Ser um Gato implica ser um Animal;
- Gato é subclass de Animal.

Uma característica do OWL-DL é que o relacionamento superclasse-subclasse pode ser computado automaticamente por um mecanismo de inferência (MI). A Figura 3.3 mostra uma representação de classes que contém indivíduos: as classes são representadas como círculos.

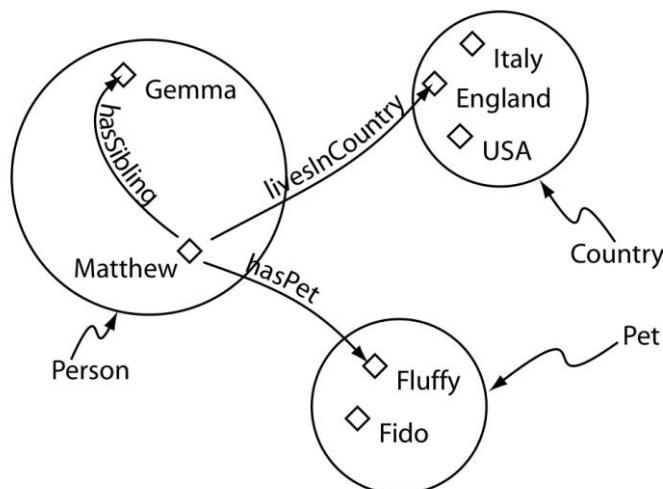


Figura 3.3 - Representação de classes contendo indivíduos

O termo *conceito* é às vezes usado no lugar de *classe*. As classes são representações concretas de conceitos.

Em OWL, as classes são construídas a partir de descrições, as quais especificam as condições que devem ser satisfeitas por um indivíduo para que ele possa ser um membro da classe. A formulação dessas descrições é explicada ao longo do tutorial.

4. Construção de uma ontologia OWL

Essa seção descreve a criação de uma ontologia de Pizzas. A ideia de utilizar pizzas reside no fato de que nesse domínio podem ser construídos bons exemplos⁴.

Exercício 2: Criar uma nova ontologia OWL no Protégé 5

1º. Abrir o editor de ontologias Protégé 5⁵.

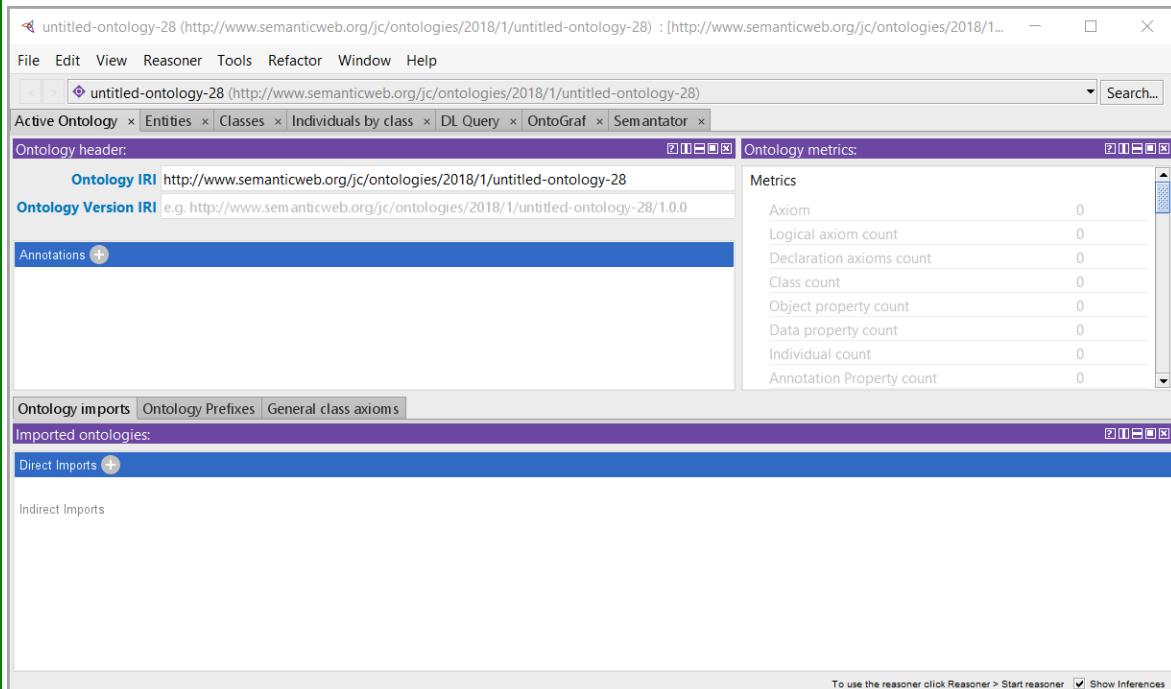


Figura 4.1 – Tela principal do editor Protégé 5

2º. Nomear e salvar o projeto:

- Criar, no espaço de trabalho (HD, Pen drive, nuvens), uma pasta: **ProjetoPtg01**
- Salvar nesta pasta, o projeto Protégé, com o seguinte nome: **OntoPizza01.owl**
- Clicar na 1ª aba: **File > Save As...** ou **[Ctrl+Shift+S]**

⁴ A ontologia que criaremos é baseada em uma Ontologia de Pizza que foi usada como base para um curso de edição de ontologias DAML + OIL em OilEd, que foi ministrado na Universidade de Manchester.

⁵ Versão usada para a elaboração deste tutorial: 5.2.0

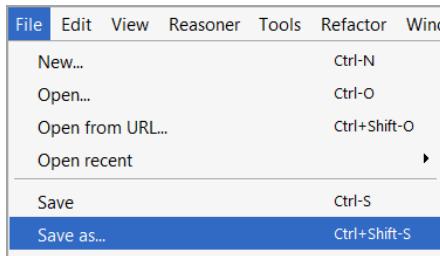


Figura 4.2

⚠ Aparece, em seguida, um menu suspenso com várias opções de formato:

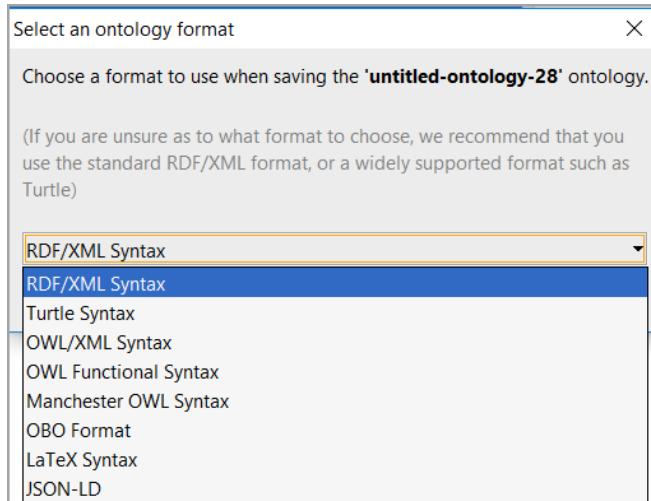


Figura 4.3

- Escolher o formato (padrão): **OWL/XML Syntax** e clicar em: **OK**
- Selecionar a pasta de armazenamento do projeto e digitar o nome do arquivo e clicar em: **Salvar**

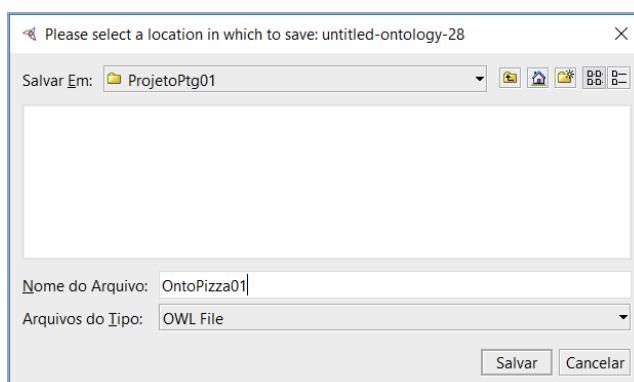


Figura 4.4

- 3º. Renomear os campos do **Ontology Header** (Cabeçalho da ontologia):

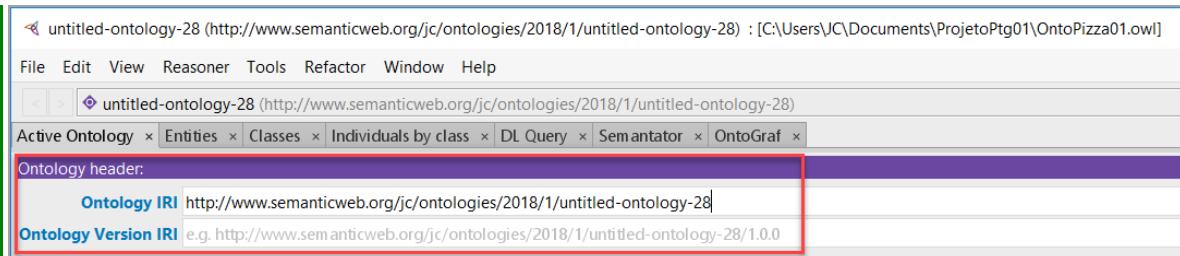


Figura 4.5

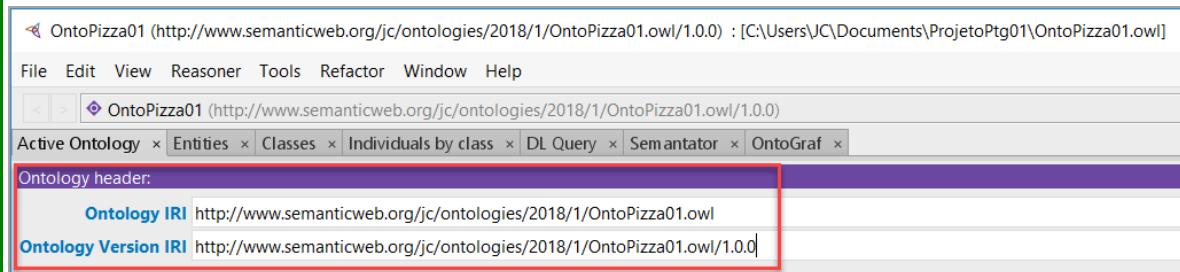


Figura 4.6

! O projeto foi devidamente criado e armazenado na pasta de trabalho.

Exercício 3: Adicionar um comentário geral sobre a ontologia criada

- 1º. Verificar que a aba **Active Ontology** (Ontologia ativa) está (ainda) selecionada.
- 2º. No painel **Annotations**, clicar no ícone para inserir comentários sobre a ontologia criada:

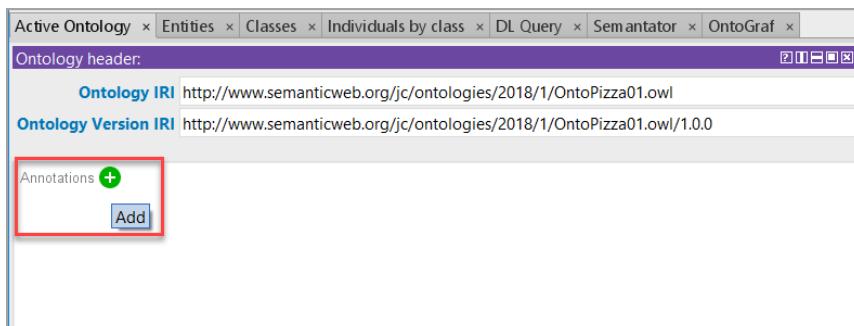


Figura 4.7

! A janela **Create Annotations** aparece (Figura 4.8, abaixo).

- 3º. Verificar que a aba **Literal** está aberta e `rdfs:comment` selecionado. Digitar o comentário: *Uma ontologia de pizza que descreve as diferentes pizzas de acordo com seus recheios.*⁶ Clicar em: **OK**

⁶ "A pizza ontology that describes various pizzas based on their toppings." (HORRIDGE; BRANDT, 2011, p. 15)

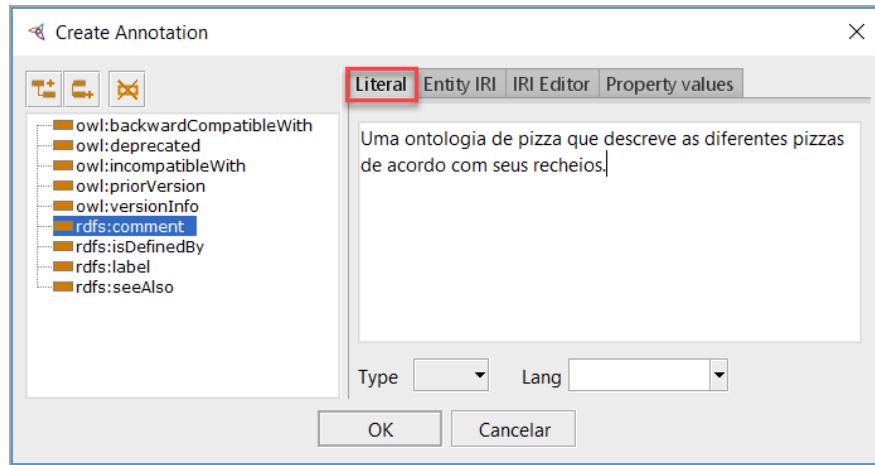


Figura 4.8

4º. Verificar na aba, **Active Ontology**, que o comentário foi devidamente gravado (quadro verde):

Figura 4.9

O comentário pode ser modificado, clicando no ícone **Edit** , ou apagado, clicando no ícone **Delete** 

4.1. Criação de classes nomeadas (Named classes)

A janela principal do Protégé é dividida em abas (*Tabs*) que apresentam as características da base de conhecimento. A aba mais importante que surge ao se iniciar um projeto é a aba *Classes*. Em geral, as classes correspondem a objetos, ou a tipos de objetos no domínio. Por exemplo, em um jornal, as classes podem ser pessoas, tais como editores, repórteres e vendedores; componentes do *layout* do jornal, tais como seções; e conteúdo do jornal, tais como anúncios e artigos.

No Protégé 5, a edição de classes é realizada usando a aba “*Classes*”, mostrada na Figura 4.10 (abaixo, no quadro vermelho). A apresentação inicial da hierarquia de classe (Class hierarchy), em estrutura de árvore, está apresentada na Figura 4.10 (faixa azul). A ontologia vazia contém apenas uma classe chamada **owl:Thing** (Coisa).

Conforme mencionado anteriormente, as classes OWL são interpretadas como conjuntos de indivíduos (ou conjuntos de objetos). A classe **owl:Thing** é a classe que representa o

conjunto que contém todos os indivíduos, uma vez que todas as classes são subclasses de **owl:Thing**⁷.

Nessa seção, o objetivo é criar classes e subclasses, modificar a hierarquia de classes, criar classes abstratas, e adicionar superclasses adicionais a classes já existentes.

Exercício 4: Criar 3 classes: Pizza, RecheioDePizza (**PizzaTopping**), MassaDePizza (**PizzaBase**)

1º. Abrir a aba **Classes**.

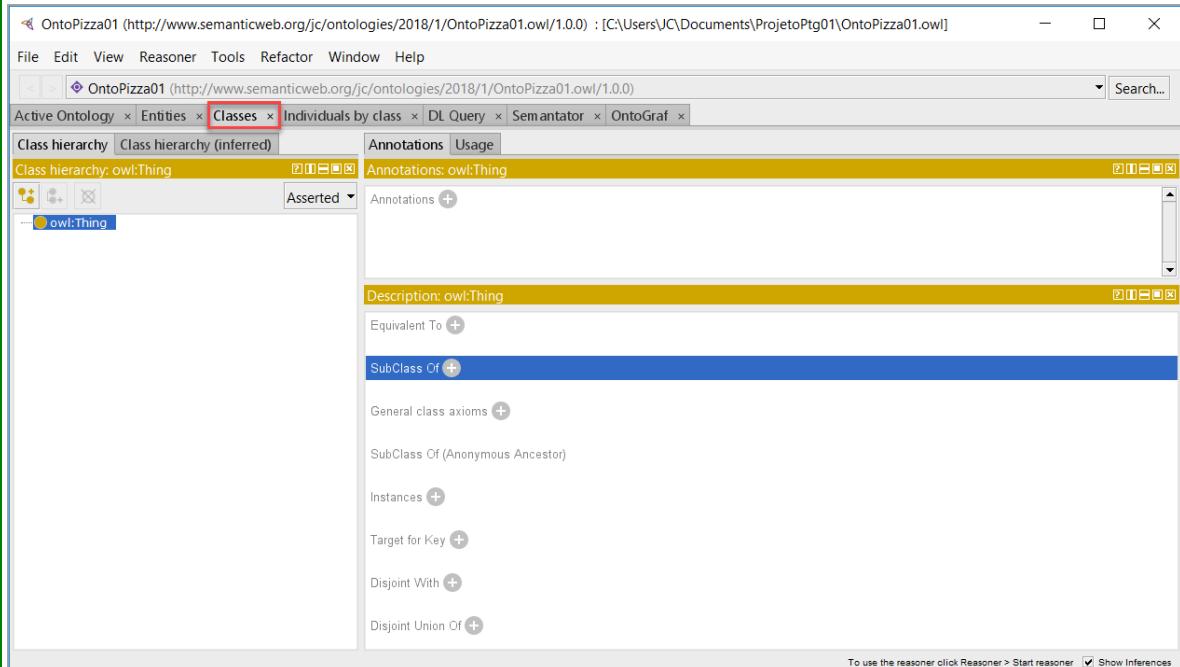


Figura 4.10

2º. Clicar no ícone **Add Subclass** para inserir uma (sub)classe:

⚠ Essa função cria uma classe, sendo uma subclasses da classe principal **owl:Thing.**

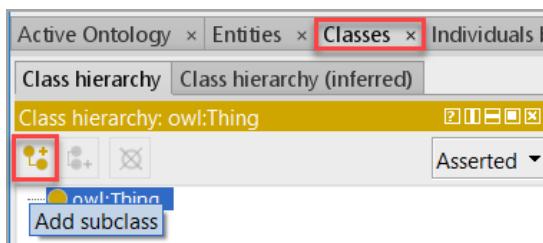


Figura 4.11

3º. Aparece uma caixa de diálogo para criar a classe **Pizza**. Digitar no campo *Name*: "Pizza" e clicar em: **OK**

⁷ "Thing" faz parte do vocabulário OWL, que é definido pela ontologia localizada em <https://www.w3.org/2002/07/owl>

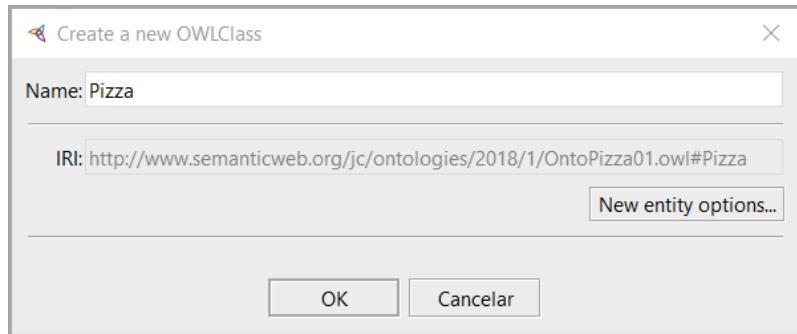


Figura 4.12

⚠ Aparece, em seguida, a janela principal com a classe criada.

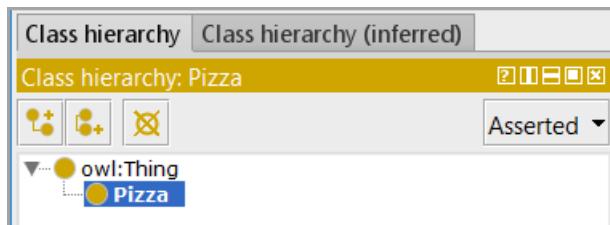


Figura 4.13

4º. **Repetir os passos anteriores** 2 e 3, para inserir as duas classes: **RecheioDePizza** (PizzaTopping), **MassaDePizza** (PizzaBase).

⚠ Observando que a classe de topo **owl:Thing deve ser selecionada (com a faixa azul - Figura 4.10, p. 20), antes de clicar em **Add subclass**, de forma que todas as classes sejam criadas como subclasses de **owl:Thing**.**

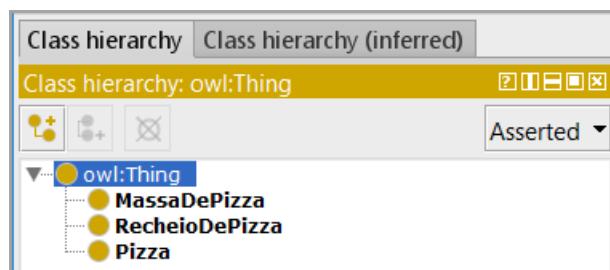


Figura 4.14

💡 Depois de criar a classe Pizza (3º passo acima), em vez de selecionar novamente owl:Thing e usar o ícone Add classe para criar RecheioDePizza e MassaDePizza como subclasses de owl:Thing, é possível usar o ícone Add sibling class (Inserir classe irmã).

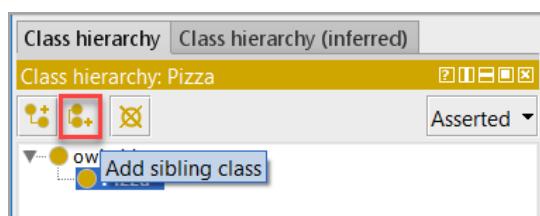


Figura 4.15

5º. [2ª opção] Selecionar, para isso, a (sub)classe **Pizza** e clicar em **Add sibling class** (Inserir classe irmã) para inserir as duas classes: **RecheioDePizza** (PizzaTopping), **MassaDePizza** (PizzaBase):

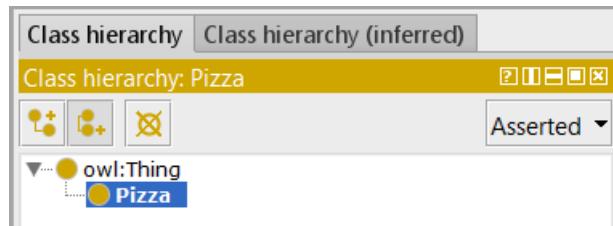


Figura 4.16

⚠ Aparece uma caixa de diálogo.

6º. Criar a classe **RecheioDePizza**. Clicar em: **OK**

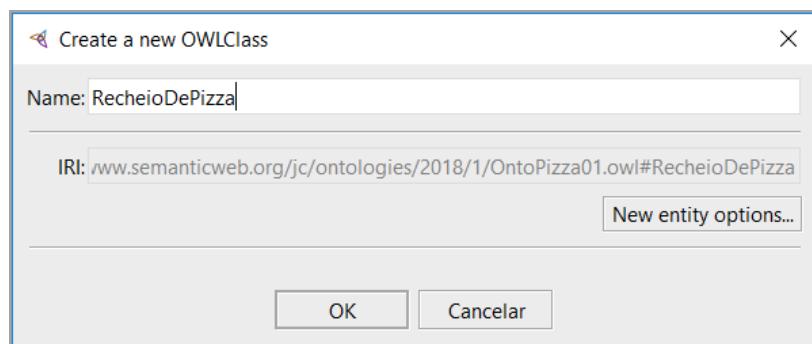


Figura 4.17

⚠ Verificar que o resultado final é parecido ao que consta na Figura 4.14, p. 21.

A hierarquia de classe também pode ser chamada de Taxonomia.



Apesar de não existir uma regra obrigatória para nomear classes *OWL*, recomenda-se que todos os nomes de classes iniciem com letra maiúscula e não contenham espaços. Este tipo de notação é conhecido como CamelBack. Por exemplo: **Pizza**, **PizzaTopping**, **MargheritaPizza**. Pode-se usar o *underscore _* para juntar palavras. Por exemplo, **Pizza_Topping**. A regra é importante para a consistência da ontologia.

4.2. Disjunção de classes

Após inserção das classes **Pizza**, **RecheioDePizza** (PizzaTopping), **MassaDePizza** (PizzaBase), é preciso agora dizer que estas classes são disjuntas, de modo que um indivíduo (individual), ou objeto, não poderá ser instância de mais de uma dentre as três classes.

Considera-se que as classes *OWL* se sobrepõem (*overlap*). Por isso, não se pode assumir que um indivíduo não é um membro de uma classe específica, simplesmente porque não se declarou que ele é um membro daquela classe. Para desconectar um grupo de classes

é preciso torná-las disjuntas. Isto garante que um indivíduo que tenha sido declarado como sendo membro de uma das classes do grupo não pode ser membro de nenhuma outra classe naquele mesmo grupo.

Exercício 5: Tornar disjuntas as classes Pizza, RecheioDePizza (PizzaTopping), MassaDePizza (PizzaBase)

1º. Verificar que (1) a aba **Classes** e, (2) a aba **Annotations** estão abertas;

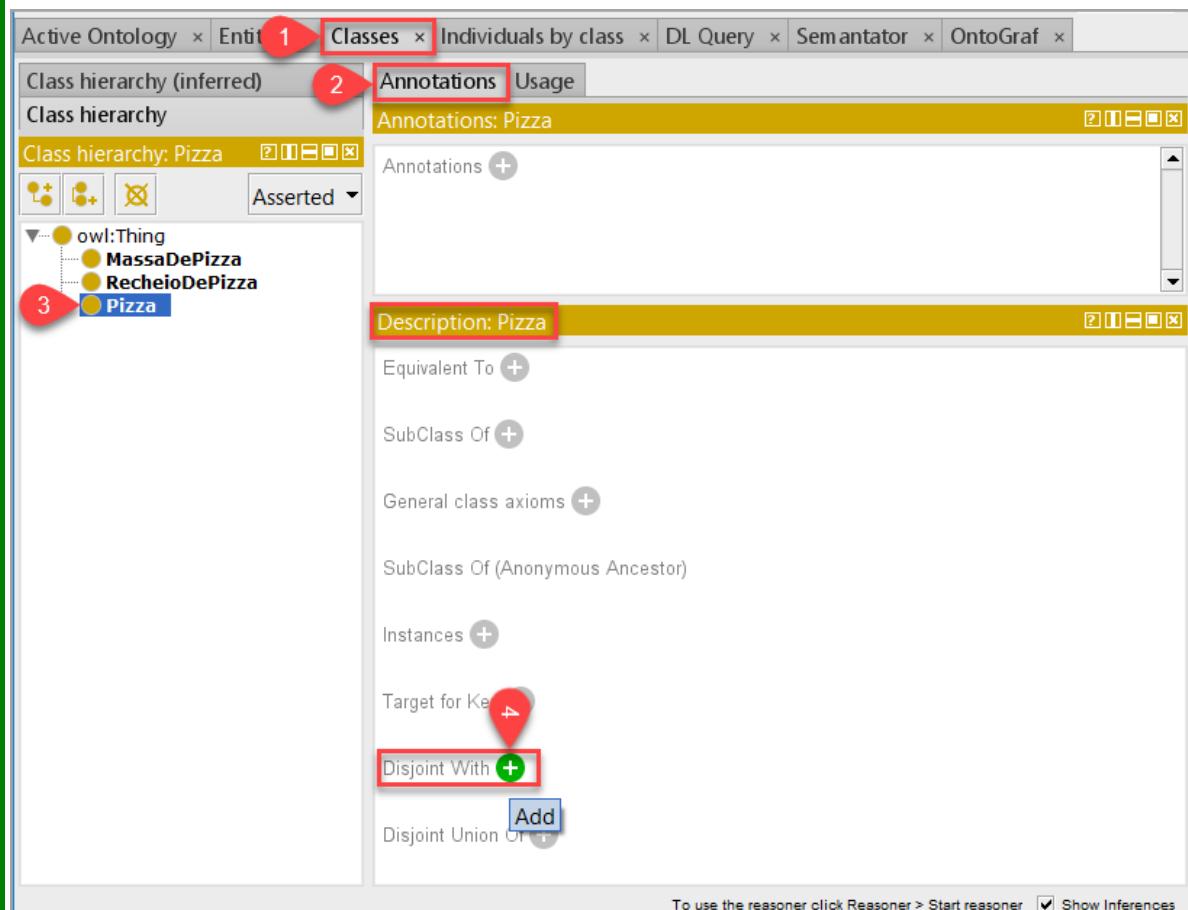


Figura 4.18

2º. Selecionar (3) a classe **Pizza**;

3º. Clicar no ícone **+ Disjoint With** (Tornar disjunto de) (4), no painel **Description** (Descrição).

⚠️ Aparece uma caixa de diálogo.

4º. Verificar que a aba **Class hierarchy** (Hierarquia de classe) está aberta. Selecionar **[Ctrl+Clique do mouse]** as classes **RecheioDePizza** (PizzaTopping) e **MassaDePizza** (PizzaBase) que devem ser disjuntas da classe **Pizza**. Clicar em: **OK**

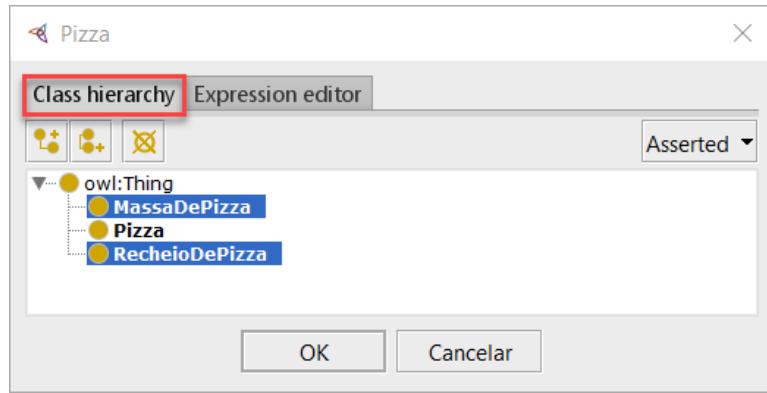


Figura 4.19

Verificar os resultados no painel **Description**:

→ A classe **Pizza** selecionada:

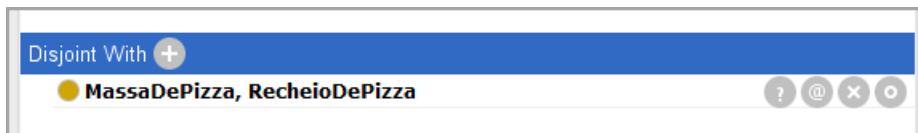


Figura 4.20

→ A classe **RecheioDePizza** (PizzaTopping) selecionada:



Figura 4.21

→ A classe **MassaDePizza** (PizzaBase) selecionada:



Figura 4.22

No exemplo, acima, a disjunção foi realizada, o que significa que não é possível a um indivíduo ser membro de uma combinação destas classes. Não faz sentido um indivíduo ser uma **Pizza** e uma **MassaDePizza** (**PizzaBase**).

4.3. Criação de classes com o assistente “Criar Hierarquia de Classe...”

Nesta seção, usaremos a ferramenta **Create Class hierarchy...** (Criar Hierarquia de Classe...) para adicionar algumas subclasses à classe **MassaDePizza** (**PizzaBase**).

Exercício 6: Usar o assistente para criar as classes **MassaFinaECrocante** (**ThinAndCrispyBase**) e **MassaGrossa** (**DeepPanBase**) como subclasses de **MassaDePizza** (**PizzaBase**)

1º. Selecionar a classe **MassaDePizza** (PizzaBase) na hierarquia de classes ① (Figura 4.23, abaixo).

2º. Abrir a aba **Tools** (Ferramentas) ②, no menu superior de Protégé, e selecionar **Create Class hierarchy...** (Criar Hierarquia de Classe...) ③:

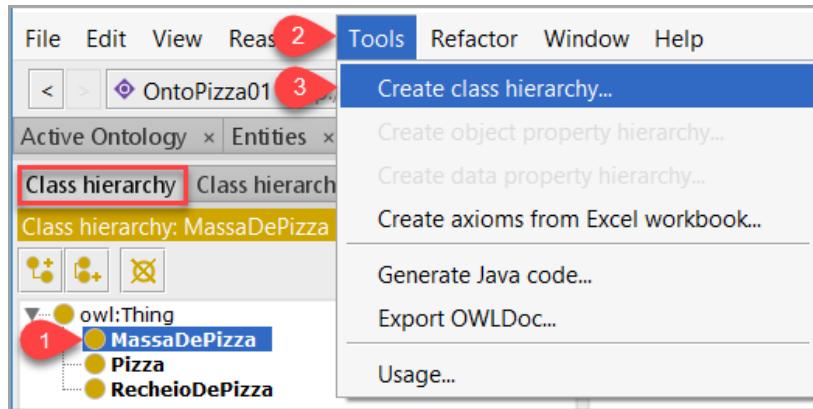


Figura 4.23

⚠️ Aparece o assistente de criação de classes.

3º. Definir as subclasses de **MassaDePizza** (PizzaBase) que devem ser criadas. Na área de texto ① (Figura 4.24, abaixo), digitar no nome da classe **MassaFinaECrocante** (ThinAndCrispyBase) e teclar **[Enter]**. Digitar o nome da classe **MassaGrossa** (DeepPanBase) e clicar em **Continue** ③ para finalizar a inserção de subclasses.

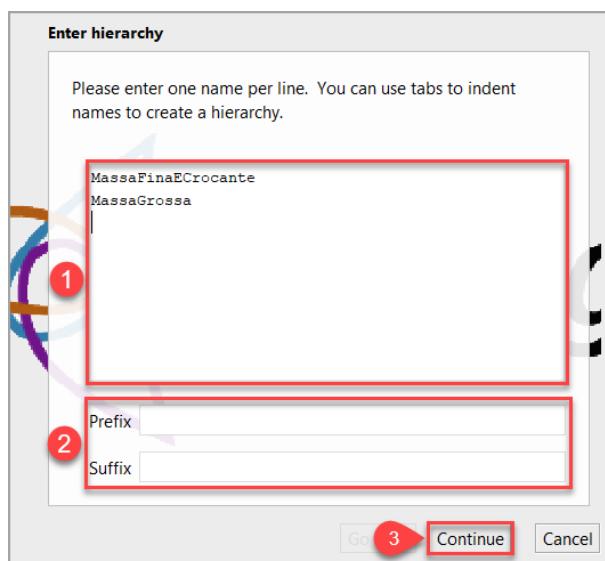


Figura 4.24

⚠️ O assistente apresenta a tela seguinte:

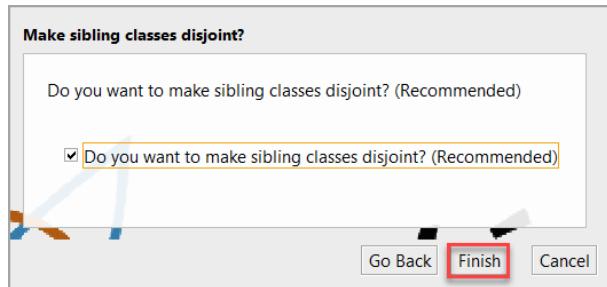


Figura 4.25

4º. Selecionar a opção **Make all primitive siblings disjoint?** (Marcar todas as primitivas irmãs como disjuntas?). Ao invés de utilizar a interface das classes disjuntas (conforme realizado no Exercício 5, p. 23), o assistente tornará as novas classes disjuntas automaticamente. Clicar em **Finish**.

⚠ Verificar o resultado gerado, no painel de hierarquia de classes, conforme a figura seguinte:

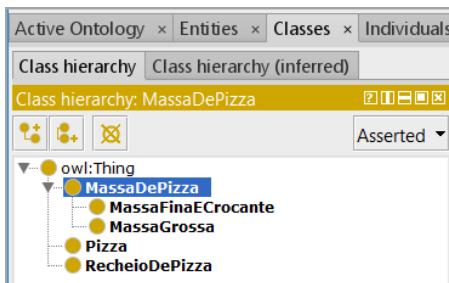


Figura 4.26

Depois de clicar em **Finish**, o assistente cria automaticamente as classes, as torna disjuntas. A ontologia tem **MassaFinaECrocante** (ThinAndCrispyBase) e **MassaGrossa** (DeepPanBase) como subclasses de PizzaBase. Essas novas classes são disjuntas e, por isso, a **MassaDePizza** (PizzaBase) não pode ser uma **MassaFinaECrocante** (ThinAndCrispyBase) e **MassaGrossa** (DeepPanBase) ao mesmo tempo.

No caso de muitas classes a adicionar, o assistente acelera o processo.

💡 Caso existam muitas classes a criar com o mesmo prefixo ou sufixo, é possível usar as opções de inserir um prefixo e/ou um sufixo, em conjunto, aos nomes de classes inseridos (ver ②, na Figura 4.24, acima).

4.3.1. Criação de algumas subclasses de RecheioDePizza (PizzaTopping)

Com algumas classes básicas inseridas, vamos criar os recheios de Pizza. Os recheios são agrupados em categorias: **RecheioDeCarne** (MeatTopping), **RecheioDeVerduras** (VegetableTopping), **RecheioDeQueijo** (CheeseTopping) e **RecheioDeFrutosDoMar** (SeafoodTopping).

Exercício 7: Criar algumas subclasses de RecheioDePizza (PizzaTopping)

- 1º. Selecionar a classe **RecheioDePizza** (PizzaTopping) na hierarquia de classes ①.
- 2º. Abrir a aba **Tools** (Ferramentas) ②, no menu superior de Protégé e selecionar **Create Class hierarchy...** (Criar Hierarquia de Classe...) ③ (Ver Figura 4.23, p.25).

Hierarquia de RecheioDePizza (PizzaTopping):	Hierarquia de PizzaTopping (original em inglês)
Queijo	Cheese
Muçarela	Mozzarella
Parmesão	Parmezan
Carne	Meat
Presunto	Ham
Calabresa	Pepperoni
Salame	Salami
CarneApimentada	SpicyBeef
FrutosDoMar	Seafood
Anchova	Anchovy
Camarão	Prawn
Atum	Tuna
Verduras	Vegetable
Alcaparras	Caper
Cogumelo	Mushroom
Azeitona	Olive
Cebola	Onion
Pimenta	Pepper
PimentaVermelha	RedPepper
PimentaVerde	GreenPepper
PimentaMexicana	JalapenoPepper
Tomate	Tomato

Quadro 4.1

- 3º. Para que todas as subclasses de **RecheioDePizza** (PizzaTopping) iniciem-se por “RecheioDe”, basta digitar no campo **Prefix** (Figura 4.24, p.25 ④) “RecheioDe” (sem as aspas). A ferramenta vai anexar automaticamente essa informação para todos os nomes das (sub)classes, evitando ter de digitá-la para cada ocorrência.
- 4º. O assistente permite que uma hierarquia de classes (ver o conteúdo acima) seja inserida usando a tecla **[Tab]** (tabulação | indentação): Digitar “Queijo” + **[Enter]** + **[Tab]** + digitar “Muçarela”, para criar uma subclasse de “Queijo”, conforme o trecho seguinte:

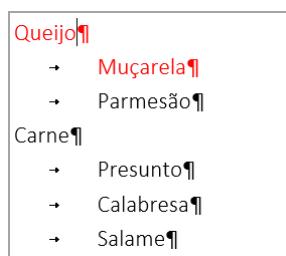


Figura 4.27

- 5º. Após digitar (ou copiar o conteúdo do arquivo .txt, da aula) a hierarquia, respeitando as indentações (recuos), deve ser parecida com a figura seguinte:

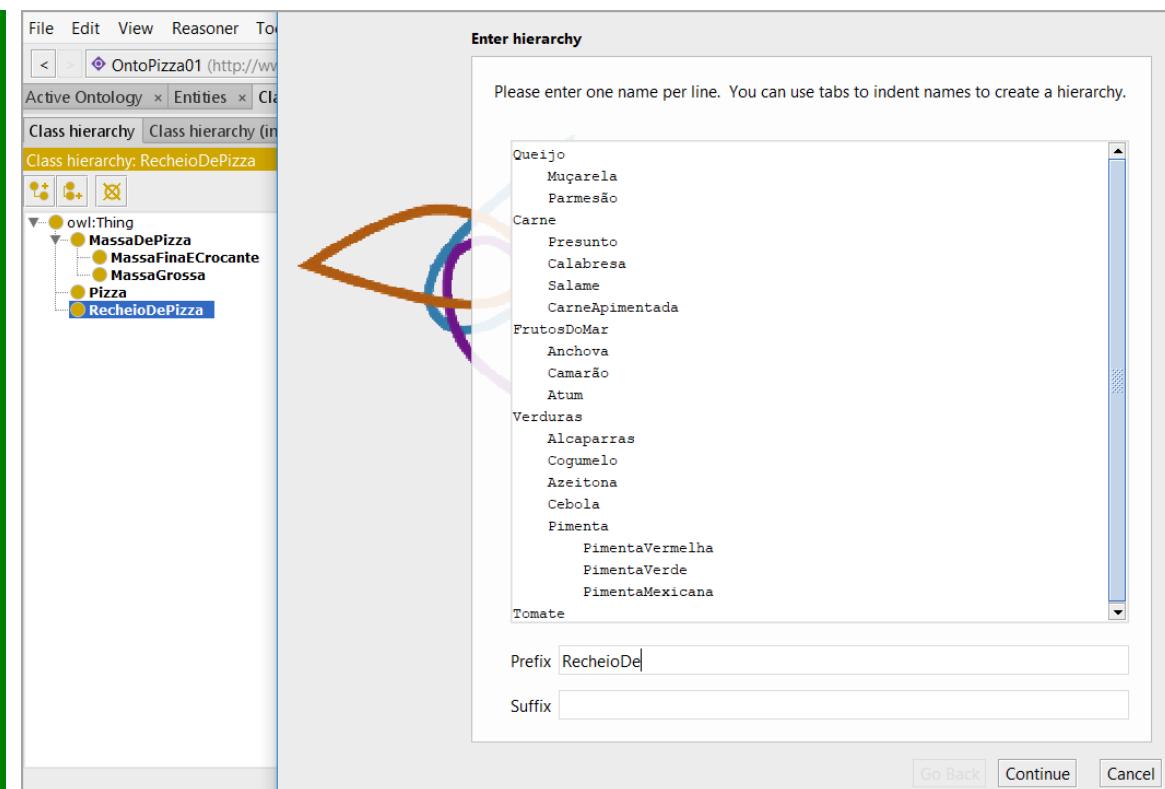


Figura 4.28

- 6º. Clicar em **Continue** para finalizar a inserção de subclasses. Selecionar a opção **Make all primitive siblings disjoint?** (Marcar todas as primitivas irmãs como disjuntas?) (Ver Figura 4.25, p.26). Clicar em **Finish** para fechar o assistente.

⚠ A nova hierarquia de classes geradas deve ser parecida com a figura seguinte. Observe-se a estrutura em árvore com as indentações (recuos) e a inserção automática do prefixo:

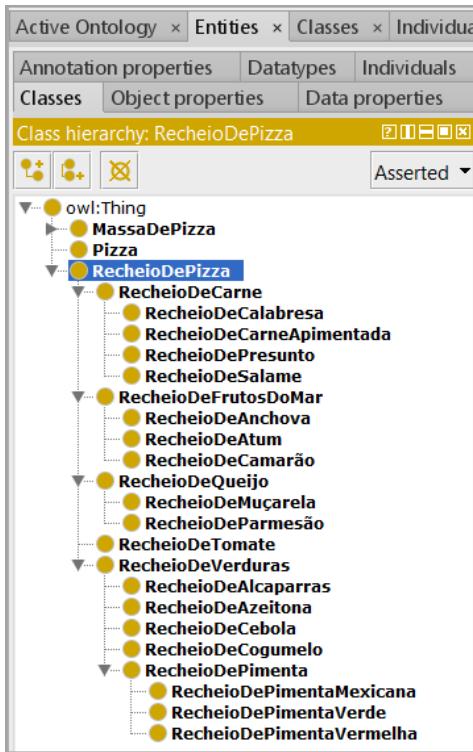


Figura 4.29

O significado de ser uma subclasse: todos os indivíduos que são membros da classe *TomatoTopping* são membros da classe *VegetableTopping* e *PizzaTopping*, uma vez que se estabeleceu que *TomatoTopping* é subclasse de *VegetableTopping*, que por sua vez é subclasse de *PizzaTopping*.

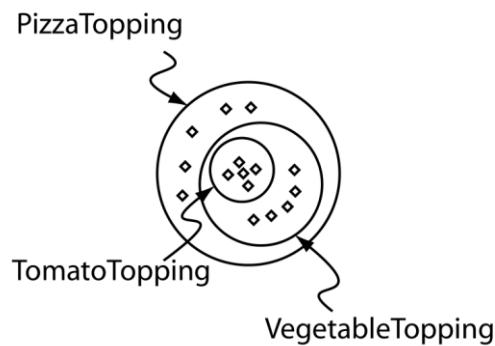


Figura 4.30 - O significado de ser uma subclasse

Até agora, foram criadas classes nomeadas simples, algumas das quais são subclasses de outras. A construção da hierarquia de classes pode parecer intuitiva. Contudo, o que realmente significa ser subclasse de alguma coisa em OWL?

Por exemplo, o que significa para **RecheioDeVerduras** (*VegetableTopping*) ser subclass de **RecheioDePizza** (*PizzaTopping*), ou para **RecheioDeTomate** (*TomatoTopping*) ser subclass de **RecheioDeVerduras** (*VegetableTopping*)? Em OWL, ser uma subclass significa uma implicação necessária. Em outras palavras, se **RecheioDeVerduras** (*VegetableTopping*) é uma subclass de **RecheioDePizza** (*PizzaTopping*) então TODAS as instâncias de

RecheioDeVerduras (VegetableTopping) são instâncias de **RecheioDePizza** (PizzaTopping), sem exceção. Se alguma coisa é um **RecheioDeVerduras** (VegetableTopping), isto implica que também é um **RecheioDePizza** (PizzaTopping), conforme apresentado na Figura 4.30 (p.29).

4.4. Propriedades em OWL

As propriedades em OWL representam relações entre dois indivíduos. Existem dois tipos principais de propriedades: *Object Properties* (Propriedades de objeto) e *DataType Properties* (Propriedades de tipo de dados). As propriedades de objeto (*Object Properties*) conectam um indivíduo a outro indivíduo. As propriedades de tipos de dados (*DataType Properties*), por sua vez, conectam um indivíduo a um valor do *XML-Schema DataType*⁸ ou a um literal do RDF (*Resource Description Framework*)⁹.

O OWL também tem um terceiro tipo de propriedade, denominada *Annotation Property* (Propriedade de Anotação), as quais são usadas para adicionar metadados às classes, aos indivíduos e às propriedades de objeto (*Object Properties*) e as propriedades de tipos de dados (*DataType Properties*).

A Figura 4.31 (abaixo) apresenta um exemplo de cada tipo de propriedade.

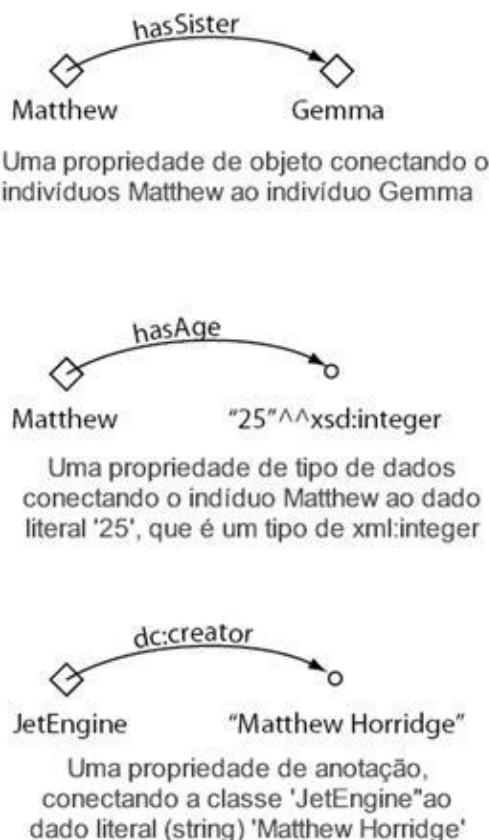


Figura 4.31 - diferentes tipos de propriedades OWL

⁸ Disponível em: <https://www.w3.org/TR/xmlschema-2/>

⁹ Disponível em: <https://www.w3.org/TR/rdf-primer/>

As propriedades podem ser criadas abrindo a aba **Object Properties** (Propriedades de Objeto), apresentada na Figura 4.32 (abaixo). Para criar propriedades OWL, clicar nos ícones destacados em ①. Como se pode ver na Figura 4.32, existem abas para criação de propriedades de tipos de dados ②, propriedades de objeto ③ e propriedades de anotação ④. A maioria das propriedades criadas neste tutorial são propriedades de objeto.

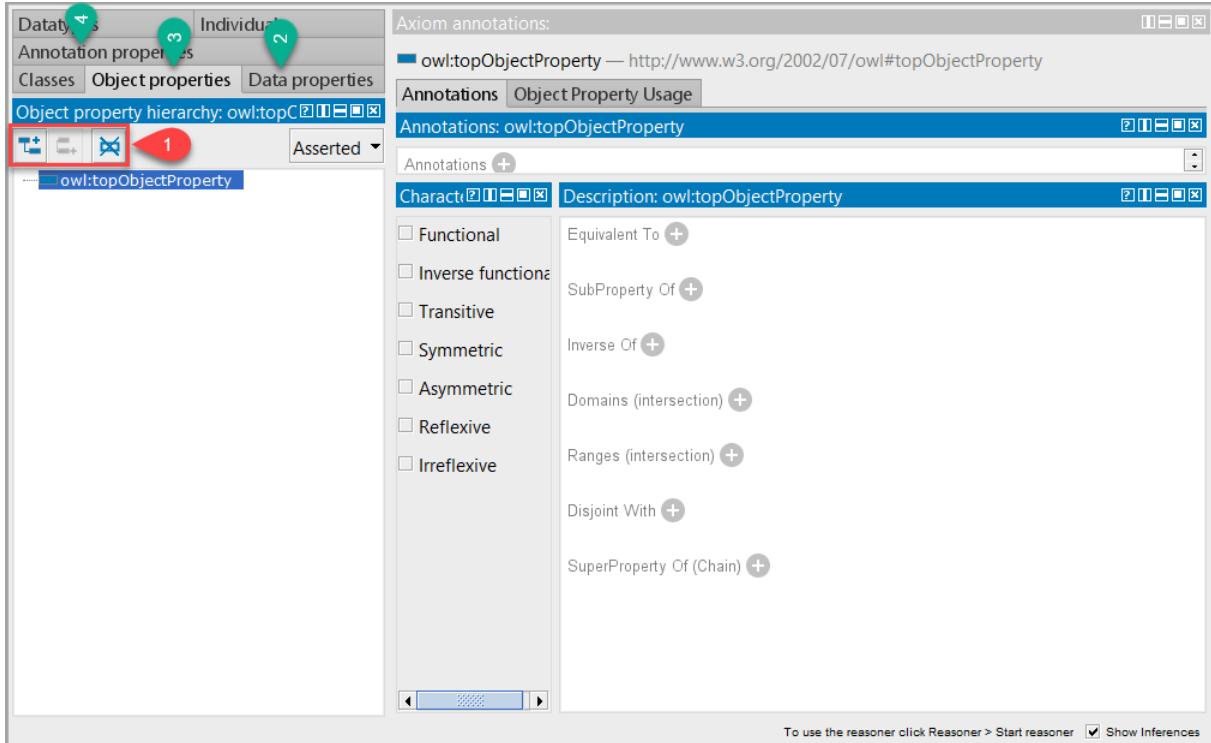


Figura 4.32 -Painel das propriedades de objeto (PropertiesTab)

 Embora não existam regras únicas para nomear propriedades, recomenda-se que os nomes das propriedades comecem com letra minúscula, sem espaço, e que a primeira letra da próxima palavra seja uma maiúscula. Recomenda-se também que as propriedades tenham como prefixo a palavra has (tem), ou a palavra is (é), por exemplo, hasPart (temParte), isPartOf (éParteDe), hasManufacturer (temFabricante), IsProducerOf (éProdutoDe).

Exercício 8: Criar uma propriedade de objeto chamada hasIngredient

1º. Abrir a aba **Object Properties** (Propriedades de objeto) ①. Clicar em **Add sub property** (Adicionar uma subpropriedade) ② para criar uma propriedade de objeto:

 O elemento “owl:topObjectProperty” deve ser selecionado (faixa azul escuro, abaixo da aba “Add sub property”, indicado pela seta verde):

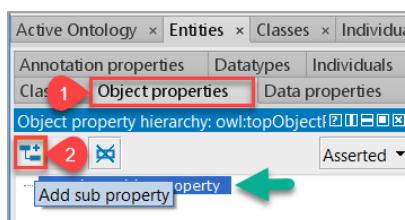


Figura 4.33

⚠ Aparece uma caixa de diálogo (parecida com a Figura 4.17, p.22)

- 2º. Inserir a propriedade **temIngredient** (**hasIngredient**) no campo “Name”, conforme apresentado na figura seguinte:

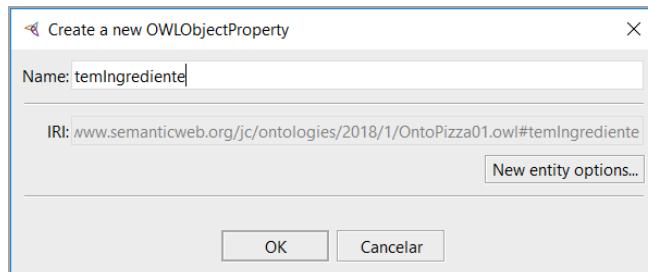


Figura 4.34

⚠ Verificar o resultado no painel “Object property hierarchy”:

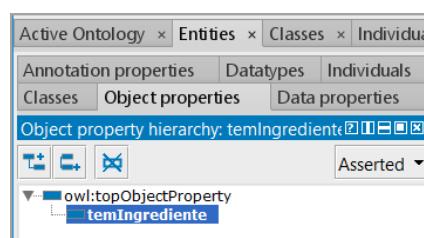


Figura 4.35

Tendo adicionado a propriedade **hasIngredient** (**temIngredient**), adicionam-se em seguida mais duas propriedades: **hasTopping** (**temRecheio**) e **hasBase** (**temBase**). As propriedades podem ter subpropriedades, de modo que se formam hierarquias de propriedades. As subpropriedades especializam superpropriedades da mesma forma que subclasses se diferenciam das superclasses.

Por exemplo, a propriedade **hasMother** (**temMãe**) é diferente da propriedade mais geral **hasParent** (**temPais**). No caso da ontologia de pizza, as propriedades **hasTopping** e **hasBase** devem ser criadas como subpropriedades de **hasIngredient** (**temIngredient**).

Se a propriedade **hasTopping** (ou **hasBase**) conecta dois indivíduos, isto implica que os dois indivíduos estão relacionados também pela propriedade **hasIngredient**.

Exercício 9: Criar duas subpropriedades de temIngredient (hasIngredient): temRecheio (hasTopping) e temMassa (hasBase)

- 1º. Para criar a propriedade **temRecheio** (**hasTopping**) como subpropriedade de **temIngredient** (**hasIngredient**), selecionar essa subpropriedade **temIngredient** (**hasIngredient**), conforme a Figura 4.35, acima (faixa azul = item selecionado).
- 2º. Clicar em **Add sub property** (Adicionar uma subpropriedade) ② (Figura 4.33, p.31) para criar **temRecheio** (**hasTopping**).

⚠ Aparece uma caixa de diálogo, idêntica com a da Figura 4.34, p. 32.

3º. Digitar **temRecheio** e clicar em **OK**

4º. Repetir a etapa anterior para criar a propriedade **temMassa** (**hasBase**): digitar **temMassa** e clicar em **OK**

⚠ Verificar a inserção das duas subpropriedades, no painel “Object property hierarchy”:

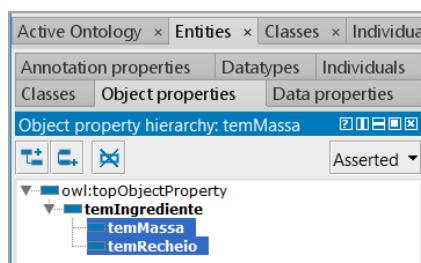


Figura 4.36

5º. **[Outra possibilidade]** O mesmo procedimento pode ser realizado de forma parecida ao do Exercício 7 (p.26), isto é, clicando com o botão direito do mouse sobre a propriedade **temIngrediente**. Aparece um menu **Add Sub-properties...**

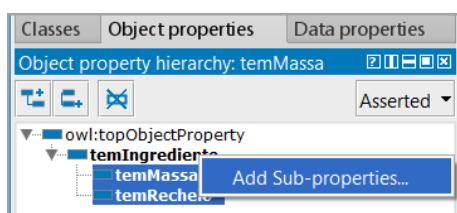


Figura 4.37

6º. Ao clicar em **Add Sub-properties...**, aparece uma caixa de diálogo, idêntica à da Figura 4.24, p.25, para digitar as duas subpropriedades **temRecheio** e **temMassa**. O resultado é idêntico ao que consta na Figura 4.36, acima.

⚠ Observação: Protégé não recomenda realizar a disjunção entre as propriedades imãs:

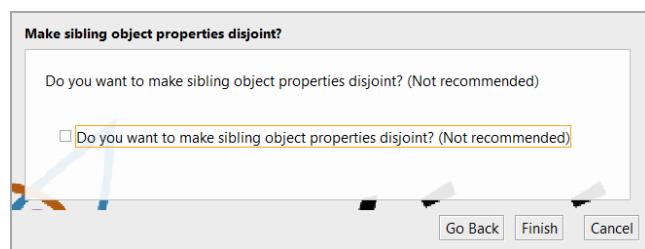


Figura 4.38

Observe que é possível criar subpropriedades de propriedades de tipos de dados. Contudo, não é possível combinar propriedades de objetos e propriedades de tipos de dados nas subpropriedades. Por exemplo, não é possível criar uma propriedade de objeto que seja a subpropriedade de uma propriedade de tipos de dados e vice-versa.

4.5. Propriedades inversas (*Inverse properties*)

Uma propriedade de objeto tem uma propriedade inversa correspondente. Se uma propriedade liga um indivíduo "A" a um indivíduo "B", então a propriedade inversa correspondente liga o indivíduo "B" ao indivíduo "A". Por exemplo, a Figura 4.39 (abaixo) mostra a propriedade **hasParent** (temPais) e sua propriedade inversa **hasChild** (temFilho): se Matthew **hasParent** (temPais) Jean, da propriedade inversa pode-se inferir que Jean **hasChild** (temFilho) Matthew.

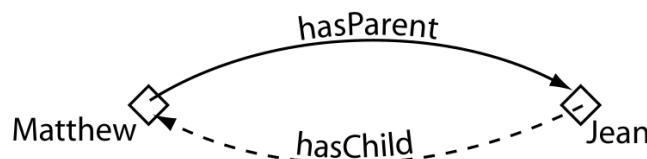


Figura 4.39 - Um exemplo de propriedade inversa

As propriedades inversas são criadas, após abertura da aba **Object Properties** (Propriedades de objeto), no campo **Inverse Of**, conforme apresentado na Figura 4.41 (p. 35).

Vamos especificar as propriedades inversas das propriedades existentes na ontologia de *Pizza*.

Exercício 10: Criar as propriedades inversas de **temIngredient**, de **temMassa**, de **temRecheio**

- 1º. Repetir os passos do Exercício 9 (anterior), para criar a propriedade **éIngredienteDe** (**isIngredientOf**), que será a propriedade inversa de **temIngredient** (**hasIngredient**), conforme apresentado na figura seguinte:

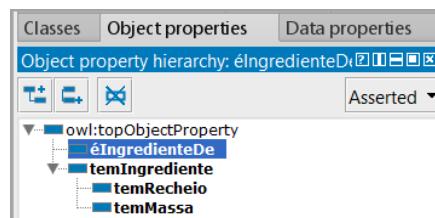


Figura 4.40

- 2º. Com a propriedade **éIngredienteDe** selecionada, clicar no ícone **+ Inverse Of** (Inverso de), no painel **Description** (Descrição), na figura seguinte:

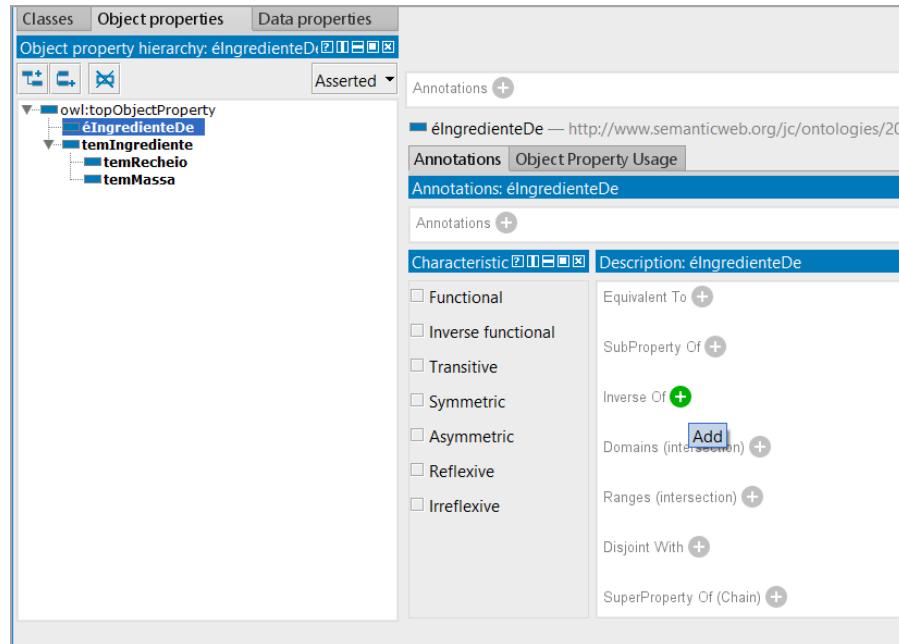


Figura 4.41

⚠ Aparece uma caixa de diálogo para selecionar a propriedade inversa temIngrediente (hasIngredient):

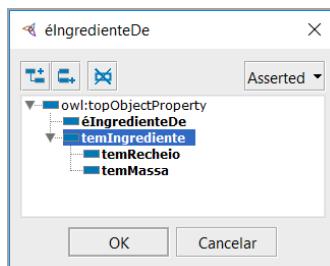


Figura 4.42

3rd. Selecionar a propriedade **temIngrediente** (hasIngredient), ou seja, propriedade inversa de **éIngredienteDe** (isIngredientOf). Clicar em **OK**.

⚠ A propriedade temIngrediente (hasIngredient) aparece agora no painel Description, em Inverse Of (Inversa de):

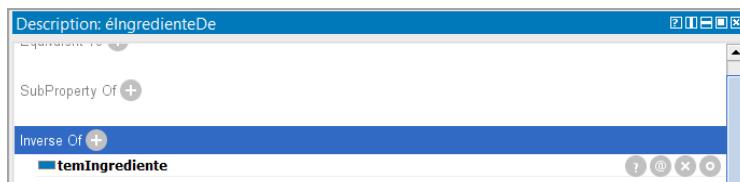


Figura 4.43

4º. Selecionar a propriedade **temMassa** (hasBase).

5º. Clicar no ícone **+ Inverse Of** (Inverso de), no painel **Description** (Descrição). Após abertura da caixa de diálogo, criar a propriedade inversa **éMassaDe** (isBaseOf) como subpropriedade de **éIngredienteDe** (isIngredientOf) e clicar em **OK**:

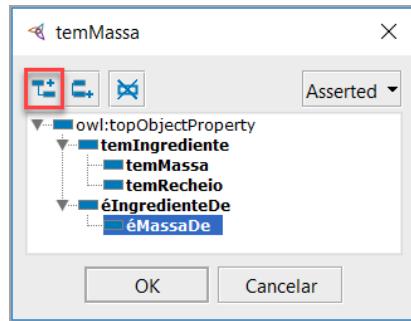


Figura 4.44

⚠ Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:



Figura 4.45

6º. Selecionar a propriedade **temRecheio** (hasTopping).

7º. Clicar no ícone **Inverse Of** (Inverso de), no painel **Description** (Descrição). Após abertura da caixa de diálogo, criar a propriedade inversa **éRecheioDe** (isToppingOf) como subpropriedade de **éIngredienteDe** (isIngredientOf) e clicar em **OK**:

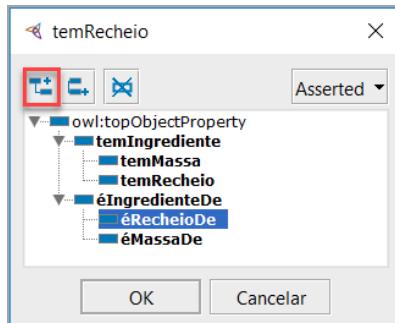


Figura 4.46

⚠ Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:



Figura 4.47

É opcional colocar as novas propriedades **éMassaDe** (isBaseOf) e **éRecheioDe** (isToppingOf) como subpropriedades de **éIngredienteDe** (isIngredientOf), conforme apresentado na Figura 4.48 (abaixo). Essas propriedades podem ser inseridas como subpropriedades de **owl:TopObjectProperty** (Figura 4.49, abaixo), pois a organização

lógica será inferida por meio do uso do mecanismo de inferência (*Reasoner*) (Figura 4.50, abaixo).

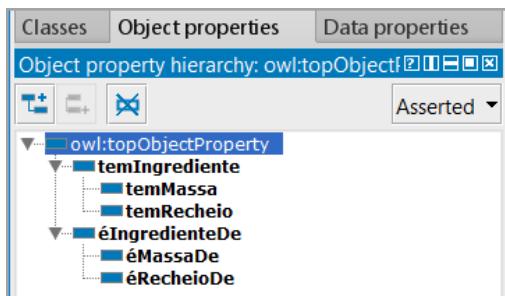


Figura 4.48

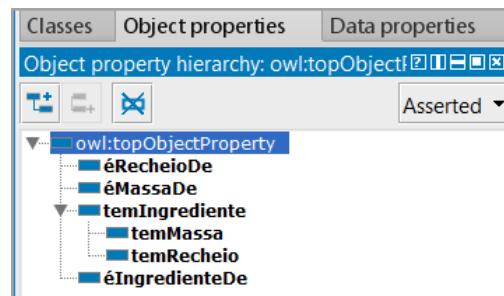


Figura 4.49

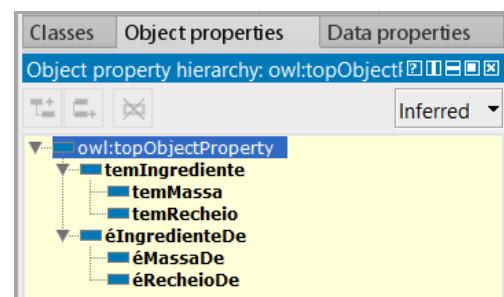


Figura 4.50

⚠ A estrutura da Figura 4.50 (*Inferred*) é idêntica à estrutura da Figura 4.48, após o processamento da estrutura da Figura 4.49 realizado com um mecanismo de inferência (*Reasoner*).

4.6. Características das propriedades de objetos (*Object Properties*)

OWL permite enriquecer o significado das propriedades, através do uso de *Property characteristics* (características das propriedades). As seções seguintes discutem as várias características que as propriedades podem ter.

4.6.1. Propriedades funcionais (*Functional properties*)

Se uma propriedade é funcional, para um determinado indivíduo “A”, pode existir até no máximo um indivíduo “B” que está relacionado ao indivíduo “A” através dessa propriedade. A Figura 4.51 (p. 38) apresenta um exemplo para a propriedade funcional **hasBirthMother** (TemMãeBiológica): alguém só pode nascer de uma única mãe.

Se **Jean hasBirthMother Peggy**, e **Jean hasBirthMother Margaret**, então **hasBirthMother** é uma propriedade funcional, ou seja, **Peggy** e **Margaret** são mesma pessoa. Contudo, observe que se **Peggy** e **Margaret** são descritas explicitamente como duas pessoas diferentes, então as afirmações anteriores levam a uma inconsistência.

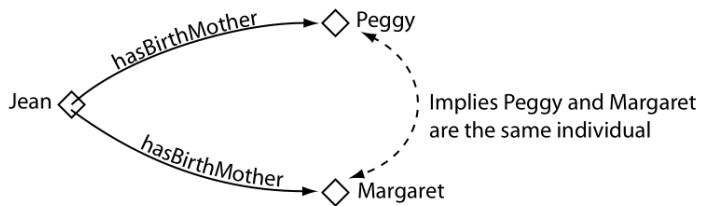


Figura 4.51 – Um exemplo de propriedade funcional

As propriedades funcionais (*Functional properties*) também são conhecidas como Single Value Properties (Propriedades de Valor Único) ou Features (Características).

4.6.2. Propriedades funcionais inversas (*Inverse Functional properties*)

Se uma propriedade é uma funcional inversa, isto significa que a sua propriedade inversa é funcional. Para o indivíduo “A”, pode existir no máximo um indivíduo relacionado ao indivíduo “A” através da propriedade.

A Figura 4.52 mostra um exemplo da propriedade funcional inversa `isBirthMotherOf` (éMãeBiológicaDe), que é a propriedade inversa de `hasBirthMother` (temMãeBiológica). Se `hasBirthMother` (temMãeBiológica) é funcional, `isBirthMotherOf` (éMãeBiológicaDe) é funcional inversa. Se **Peggy** é a mãe natural de **Jean**, e **Margaret** é a mãe natural de **Jean**, infere-se que **Peggy** e **Margaret** correspondem à mesma pessoa.

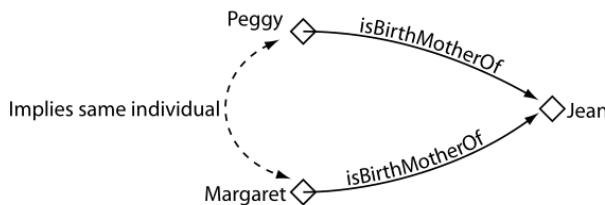


Figura 4.52 - Exemplo de uma propriedade funcional inversa

4.6.3. Propriedades transitivas (*Transitive Properties*)

Se uma propriedade P *transitiva* relaciona o indivíduo “A” ao indivíduo “B”, e também um indivíduo “B” ao indivíduo “C”, infere-se que o indivíduo “A” está relacionado ao indivíduo “C” através da propriedade P.

Por exemplo, a Figura 4.53 mostra um exemplo da propriedade transitiva `hasAncestor` (temAncestral). Se o indivíduo **Matthew** tem o ancestral **Peter**, e **Peter** tem o ancestral **William**, então **Matthew** tem um ancestral que é **William**. Esse fato é indicado pela linha tracejada na Figura 4.53.

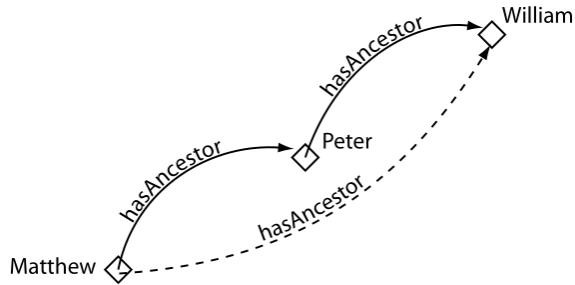


Figura 4.53 – Um exemplo de propriedade transitiva

4.6.4. Propriedades simétricas (*Symmetric properties*)

Se uma propriedade P é *simétrica*, e relaciona um indivíduo "A" ao indivíduo "B", então o indivíduo "B" também está relacionado ao indivíduo "A" através da propriedade P.

A Figura 4.54 mostra um exemplo. Se o indivíduo **Matthew** está relacionado ao indivíduo **Gemma** através da propriedade **hasSibling** (temIrmão), então **Gemma** também está relacionada a **Matthew** através da propriedade **hasSibling**. Em outras palavras, se **Matthew** tem uma irmã **Gemma**, então **Gemma** tem um irmão que é **Matthew**. Dito de outra forma, a propriedade é a própria inversa.



Figura 4.54 – Um exemplo de propriedade simétrica

Deseja-se tornar **temIngredient** (**hasIngredient**) uma propriedade transitiva, de modo que, por exemplo, se um recheio de pizza tem um ingrediente, então uma pizza do mesmo recheio deve ter o mesmo ingrediente.

Para definir as características da propriedade, utiliza-se o campo de *Property characteristics* (Características da Propriedade) conforme a Figura 4.55, a qual está localizada na aba **Object Properties** (Propriedades de objeto), no painel **Property characteristics** (Características da Propriedade).

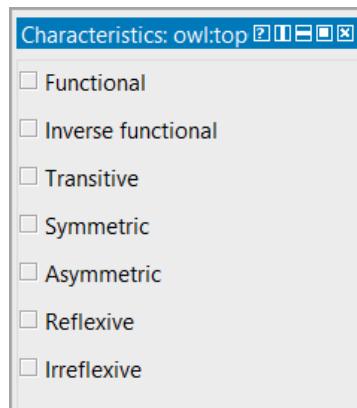


Figura 4.55

Exercício 11: Tornar transitiva a propriedade temIngredient (hasIngredient)

- 1º. Na aba **Object Properties** (Propriedades de objeto), selecionar a propriedade **temIngredient** (hasIngredient) na hierarquia de propriedades.
- 2º. Marcar a opção **Transitive** no painel **Property characteristics** (Características da Propriedade) (Figura 4.56, abaixo).
- 3º. Selecionar a propriedade **éIngredienteDe** (isIngredientOf), que é a inversa de **temIngredient** (hasIngredient). Marcar a opção **Transitive** no painel **Characteristic** (Característica) (Figura 4.57, abaixo).

The screenshot shows the Protégé interface with the 'Object properties' tab selected. In the main pane, the 'Object property hierarchy: temIngredient' is displayed, showing 'temMassa' and 'temRecheio' as sub-properties of 'temIngredient'. Below this, 'éIngredienteDe' is shown as the inverse property. The right-hand panel shows the 'Characteristic' section for 'temIngredient', with the 'Transitive' checkbox checked (indicated by a red circle with '1'). Other options like 'Functional', 'Inverse functional', 'Symmetric', 'Asymmetric', 'Reflexive', and 'Irreflexive' are also listed.

Figura 4.56

The screenshot shows the Protégé interface with the 'Object properties' tab selected. In the main pane, the 'Object property hierarchy: éIngredienteDe' is displayed, showing 'temMassa' and 'temRecheio' as sub-properties of 'éIngredienteDe'. Below this, 'temIngredient' is shown as the inverse property. The right-hand panel shows the 'Characteristic' section for 'éIngredienteDe', with the 'Transitive' checkbox checked (indicated by a red circle with '1'). Other options like 'Functional', 'Inverse functional', 'Symmetric', 'Asymmetric', 'Reflexive', and 'Irreflexive' are also listed.

Figura 4.57

💡 Se uma propriedade é transitiva, então a propriedade inversa a ela também é transitiva. **No momento, essa operação ainda é feita manualmente no Protégé 5** (Figura 4.57, acima). Contudo, o mecanismo de inferência (*Reasoner*) assume que se a propriedade é transitiva, a propriedade inversa também é transitiva.

⚠️ Se uma propriedade é transitiva ela não pode ser funcional, uma vez que a propriedade transitiva, por sua própria natureza, pode formar cadeias de indivíduos. **Tornar uma propriedade transitiva funcional, portanto, não faz sentido.**

Objetiva-se agora dizer que uma pizza pode ter apenas um tipo de massa (*Base*). Existem várias formas para fazer isso. Escolhe-se tornar **temMassa** (hasBase) uma propriedade funcional, de modo que ela possa ter apenas um valor para um determinado indivíduo.

Exercício 12: Tornar funcional a propriedade temMassa (hasBase)

- 1º. Na aba **Object Properties** (Propriedades de objeto), selecionar a propriedade **temMassa** (hasBase).

2º. Marcar a opção **Functional** no painel **Property characteristics** (Características da Propriedade):

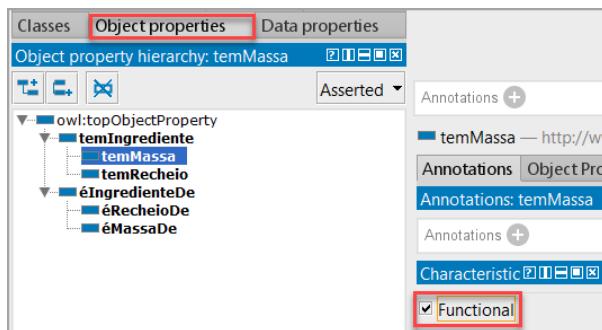


Figura 4.58

4.6.5. Propriedades assimétricas (*Asymmetric properties*)¹⁰

Se uma propriedade P é assimétrica e a propriedade relaciona o indivíduo “A” ao indivíduo “B”, então o indivíduo “B” não pode ser relacionado ao indivíduo “A” por meio de uma propriedade individual P. A Figura 4.59 mostra um exemplo de uma propriedade assimétrica.

Se o indivíduo **Jean** estiver relacionado ao indivíduo **Matthew** através da propriedade **hasChild**, pode-se inferir que Matthew não está relacionado com Jean através da propriedade **hasChild**.¹¹

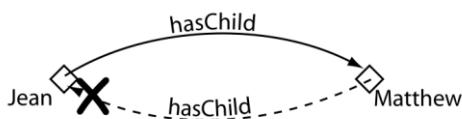


Figura 4.59 – Um exemplo de propriedade assimétrica

4.6.6. Propriedades reflexivas (*Reflexive properties*)¹²

A propriedade P é considerada reflexiva quando ela relaciona o indivíduo “A” com ele mesmo.

Na Figura 4.60, podemos ver um exemplo de uso da propriedade **knows** (conhece): um indivíduo **George** deve ter uma relação consigo mesmo, usando a propriedade **knows** (conhece). Em outras palavras, **George** deve conhecer a si mesmo. No entanto, além disso, é possível que **George** conheça outras pessoas; portanto, o indivíduo **George** pode ter uma relação com o indivíduo **Simon** por meio da propriedade **knows** (conhece).

¹⁰ Esse item consta na versão 1.3. (HORRIDGE; BRANDT, 2011), não consta na versão 1.0 (HORRIDGE et al., 2004).

¹¹ Esse parágrafo foi adaptado para se adequar com a ilustração (fig. 4.23) associada, na versão 1.3 (HORRIDGE; BRANDT, 2011, p. 32).

¹² Esse item consta na versão 1.3. (HORRIDGE; BRANDT, 2011), não consta na versão 1.0 (HORRIDGE et al., 2004).

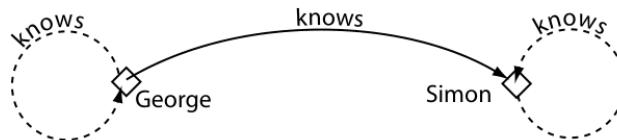


Figura 4.60 – Um exemplo de propriedade reflexiva

4.6.7. Propriedades irreflexivas (*Irreflexive properties*)¹³

Se uma propriedade P é irreflexiva, ela pode ser descrita como uma propriedade que relaciona um indivíduo "A" a um indivíduo "B", onde o indivíduo "A" e o indivíduo "B" não são os mesmos.

Um exemplo disso seria a propriedade da **isMotherOf** (éMãeDe). De um indivíduo, Alice pode estar relacionada com o indivíduo Bob por meio da propriedade **isMotherOf** (éMãeDe), mas Alice não pode ser mãe (**isMotherOf**) de si mesma (Figura 4.61).

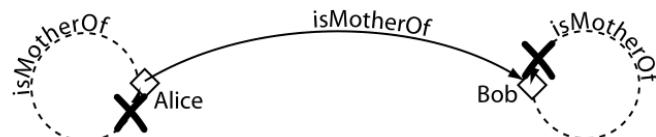


Figura 4.61 - Um exemplo de propriedade irreflexiva

4.7. Domínios (*Domains*) e Escopos (*Ranges*) de uma propriedade

Uma propriedade possui um *domain* (domínio) e um *range* (escopo). As propriedades conectam indivíduos de um domínio (*domain*) a indivíduos de um escopo (*range*).

Por exemplo, na ontologia de *Pizza*, a propriedade **hasTopping** (temRecheio) liga indivíduos pertencentes à classe **Pizza** a indivíduos pertencentes à classe **PizzaTopping** (RecheioDePizza).

Neste caso, o domínio (*domain*) da propriedade **hasTopping** é **Pizza** e o escopo (*range*) é **PizzaTopping** (RecheioDePizza), conforme apresentado na Figura 4.62: o domínio (*domain*) e o escopo (*range*) para a propriedade **hasTopping** (temRecheio) e suas propriedades inversas **isToppingOf** (éRecheioDe).

O domínio (*domain*) para **hasTopping** é **Pizza** e o escopo (*range*) para **hasTopping** é **PizzaTopping** (RecheioDePizza). O *domain* e o escopo (*range*) para **isToppingOf** são o domínio (*domain*) e o escopo (*range*) para **hasTopping**.

¹³ Esse item consta na versão 1.3. (HORRIDGE; BRANDT, 2011), não consta na versão 1.0 (HORRIDGE et al., 2004).

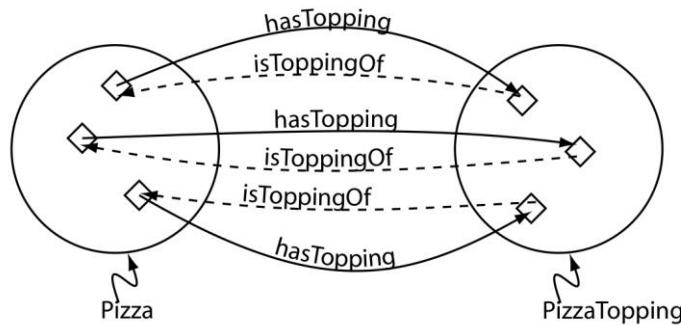


Figura 4.62 – Um exemplo de Domínio e de Escopo

⚠ Domains (domínios) e Ranges (escopos) em OWL não são restrições sujeitas à verificação, são utilizados como axiomas em inferências. Por exemplo, se a propriedade `hasTopping` tem o conjunto domínio `Pizza` e aplica-se a propriedade `hasTopping` a `IceCream` (indivíduos membros da classe `IceCream`), o resultado pode ser um erro. É possível inferir que a classe `IceCream` é subclasse de `Pizza` (um erro é gerado através do mecanismo de inferência (MI), apenas se `Pizza` for disjunta de `IceCream`).

Deseja-se especificar que a propriedade `temRecheio` (`hasTopping`) tem um escopo (range) `RecheioDePizza` (`PizzaTopping`). Para tal, usa-se a função **Ranges** (Figura 4.63), disponível, no Protégé 5, no painel **Object Property** (Propriedade de objeto).

Exercício 13: Especificar a classe `RecheioDePizza` (`PizzaTopping`) como o escopo (Range) da propriedade `temRecheio` (`hasTopping`)

- 1º. Na aba **Object Property** (Propriedade de objeto), selecionar a propriedade `temRecheio` (`hasTopping`).
- 2º. Clicar no ícone **+ Ranges** (Escopos), no painel **Properties Description** (Descrição de propriedades).

Figura 4.63

3º. Após abertura da caixa de diálogo, selecionar a classe **RecheioDePizza** (PizzaTopping) na aba **Class hierarchy** (Hierarquia de classe) e clicar em **OK**:

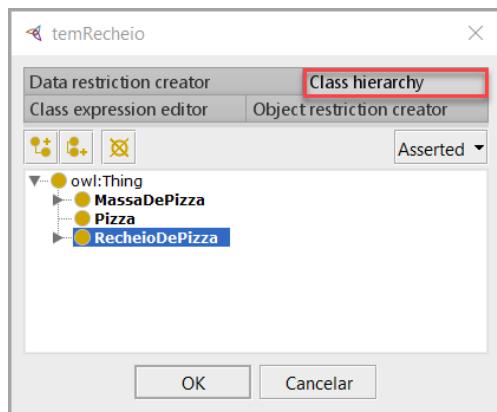


Figura 4.64

⚠ Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:



Figura 4.65

Também é possível, mas não recomendável, indicar que uma classe e não seus indivíduos são escopo de uma propriedade. É erro pensar que o escopo de uma propriedade é uma classe, quando um escopo corresponde na verdade aos indivíduos membros da classe. Ao especificar o escopo de uma propriedade como uma classe, trata-se tal classe como um indivíduo. Isto é um tipo de metadeclaração, e pode levar a ontologia para o OWL-Full. Lembrando que as instâncias de propriedades de objeto ligam indivíduos a indivíduos.¹⁴

É possível especificar várias classes como escopo de uma propriedade. Caso isso seja feito no *Protégé 5*, o escopo da propriedade é interpretado como uma união das classes. Por exemplo, se uma propriedade tem as classes **Man** (homem) e **Woman** (mulher) listadas na interface **Ranges**, isso significa que o escopo daquela propriedade será interpretada como **Man** união com **Woman**

Exercício 14: Especificar Pizza o domínio (**Domain**) da propriedade temRecheio (**hasTopping**)

1º. Na aba **Object Properties** (Propriedades de objeto), selecionar a propriedade **temRecheio** (**hasTopping**).

2º. Clicar no ícone **+ Domains** (Domínios), no painel **Properties Description** (Descrição de propriedades):

¹⁴ Essa observação consta na versão 1.0 (HORRIDGE et al., 2004, p. 36), mas foi excluída da versão 1.3 (HORRIDGE; BRANDT, 2011).

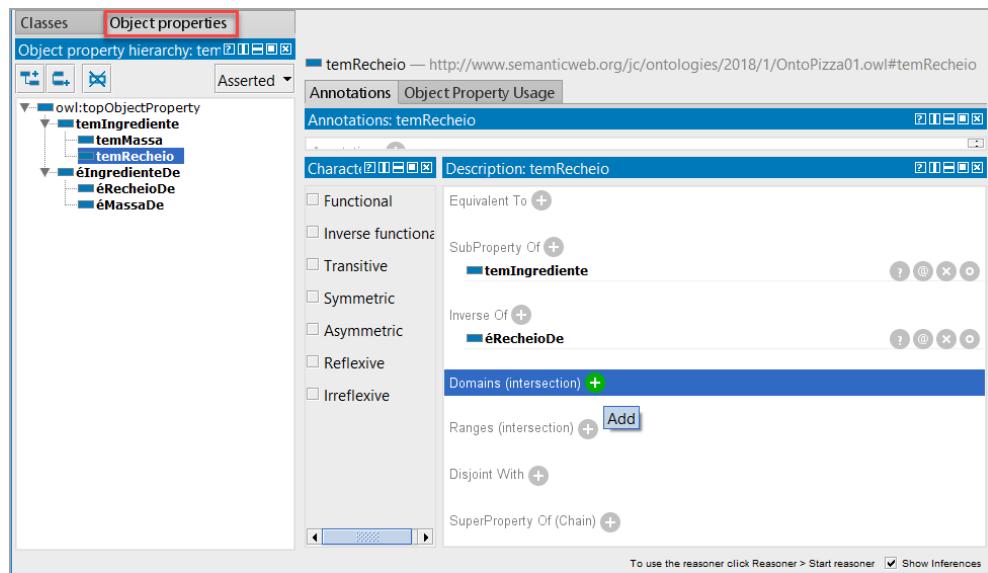


Figura 4.66

3º. Após abertura da caixa de diálogo, selecionar a classe **Pizza** na aba **Class hierarchy** (Hierarquia de classe) e clicar em **OK**:

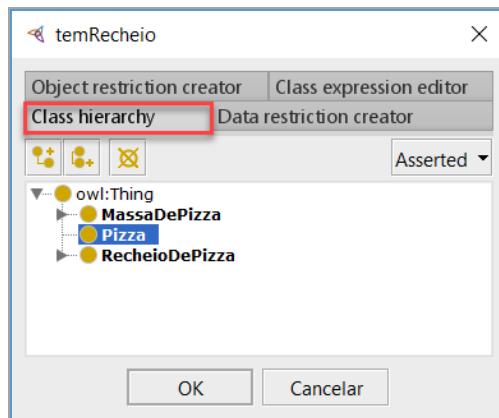


Figura 4.67

⚠ Verificar o resultado gerado, no painel **Description, conforme a figura seguinte:**

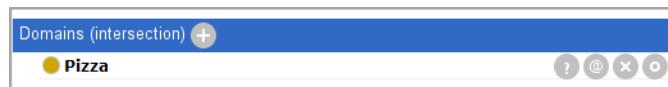


Figura 4.68

Os indivíduos citados "do lado esquerdo" (Figura 4.62, p. 43) da propriedade **temRecheio** (hasTopping) são membros da classe **Pizza**; os indivíduos citados "do lado direito" da propriedade **temRecheio** (hasTopping) são membros da classe **RecheioDePizza** (PizzaTopping).

Por exemplo, sejam os indivíduos "A" e "B" e uma declaração "**A hasTopping B**": infere-se que "A" é um membro da classe Pizza e que "B" é um membro da classe PizzaTopping¹⁵.

⚠ Quando diversas classes são especificadas como *domain* (domínio) de uma propriedade, o Protégé interpreta o *domain* (domínio) da propriedade como a união dessas classes.¹⁶

⚠ Embora a linguagem OWL permita o uso de *class expressions* (expressões de classes) subjetivas para o *domain* (domínio) de uma propriedade, isto não é permitido durante a edição de ontologias no Protégé.¹⁷

Deseja-se especificar o *domínio* (*domain*) e o *escopo* (*range*) da propriedade **éRecheioDe** (*isToppingOf*), sendo a propriedade inversa de **temRecheio** (*hasTopping*).

Exercício 14b¹⁸: Especificar o domínio (*domain*) e o escopo (*range*) da propriedade **éRecheioDe** (*isToppingOf*)

- 1º. Na aba **Object Properties** (Propriedades de objeto), selecionar a propriedade **éRecheioDe** (*isToppingOf*).
- 2º. Definir a classe **RecheioDePizza** (PizzaTopping) como o domínio da propriedade **éRecheioDe** (*isToppingOf*).
- 3º. Definir a classe **Pizza** como o escopo da propriedade **éRecheioDe** (*isToppingOf*).

⚠ Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:



Figura 4.69

Observação¹⁹: Observe-se na propriedade **éRecheioDe** (*isToppingOf*) — propriedade inversa do **temRecheio** (*hasTopping*) —, que Protégé preencheu automaticamente o domínio e o escopo da propriedade **éRecheioDe** (*isToppingOf*) porque o domínio e o escopo da propriedade inversa foram especificados. O escopo de **éRecheioDe** (*isToppingOf*) é o

¹⁵ Isso seria o caso, mesmo que "A" não tenha sido declarado membro da classe Pizza e/ou "B" não tenha sido declarado membro da classe PizzaTopping.

¹⁶ Essa observação consta na versão 1.0 (HORRIDGE et al., 2004, p. 37), mas foi excluída da versão 1.3 (HORRIDGE; BRANDT, 2011).

¹⁷ Essa observação consta na versão 1.0 (HORRIDGE et al., 2004, p. 37), mas foi excluída da versão 1.3 (HORRIDGE; BRANDT, 2011).

¹⁸ Esse exercício corresponde ao Exercício 14 da versão 1.0 (HORRIDGE et al., 2004, p. 38), ele foi excluído da versão 1.3 (HORRIDGE; BRANDT, 2011) e substituído por uma observação (encontra-se na sequência do Exercício 14b, após a Figura 4.69, p. 43).

¹⁹ Nota JCM: Essa nota consta na versão 1.3 (HORRIDGE; BRANDT, 2011, p. 36). Assim, a criação automática, por Protégé, de domínio e de escopo de propriedades inversas, justificaria, a meu ver, a exclusão do exercício 14 da versão 1.0 (ver a nota de rodapé n. 18). Não entanto, essa automatização não foi conclusiva no Protégé 5, até com MI. Portanto, resolvi manter esse exercício (renomeado 14b).

domínio da propriedade inversa **temRecheio** (hasTopping) e o domínio de **éRecheioDe** (isToppingOf) é o escopo da propriedade inversa **temRecheio** (hasTopping).

Exercício 15: Especificar o domínio (Domain) e o escopo (Range) da propriedade **temMassa (hasBase) e de sua propriedade inversa **éMassaDe** (isBaseOf)**

- 4º. Na aba **Object Properties** (Propriedades de objeto), selecionar a propriedade **temMassa** (hasBase).
- 5º. Especificar a classe **Pizza** como o domínio (Domain) da propriedade **temMassa** (hasBase).
- 6º. Especificar a classe **MassaDePizza** (PizzaBase) como o escopo (Range) da propriedade **temMassa** (hasBase).
- 7º. Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:

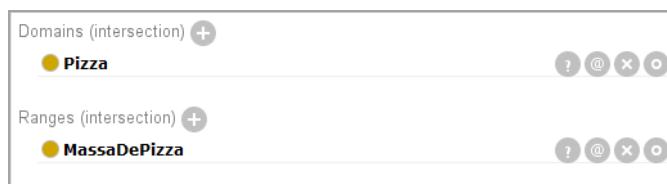


Figura 4.70

- 8º. Selecionar a propriedade **éMassaDe** (isBaseOf).
- 9º. Definir **MassaDePizza** (PizzaBase) como o domínio da propriedade **éMassaDe** (isBaseOf).
- 10º. Definir **Pizza** como o escopo da propriedade **éMassaDe** (isBaseOf).

⚠️ Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:



Figura 4.71

É preciso garantir que o domínio e o escopo para as propriedades estão também configurados para as propriedades inversas de maneira correta. Em geral, o domínio para uma propriedade é o escopo de seu inverso, e o escopo para uma propriedade é o domínio de sua inversa (conforme apresentado na Figura 4.62, p. 43).

⚠️ Embora se tenha especificado domínios e escopos de várias propriedades para o presente tutorial, **não se recomenda que esse procedimento seja rotineiro**. As condições de domínio e de escopo não se comportam como restrições e, além disso, podem causar resultados inesperados na classificação. Esses problemas e seus efeitos indesejados são de difícil localização em uma grande ontologia.

4.8. Descrição e definição de classes

Após a criação de algumas propriedades pode-se agora utilizá-las para definir e descrever as classes da ontologia de Pizza.

4.8.1. Restrições de propriedades

Lembre-se de que, em OWL, as propriedades descrevem relações binárias. As propriedades de tipo de dados (*Data Type Properties*) descrevem relações entre indivíduos e valores de dados. As propriedades do objeto (*Object Properties*) descrevem as relações entre dois indivíduos.

Por exemplo, na Figura 3.2 (p. 15), o indivíduo **Mateus** está relacionado ao indivíduo **Gemma** através da propriedade **hasSibling**. Agora, considere-se todos os indivíduos que possuem um tipo de relação com outros indivíduos. Assim, esses indivíduos pertencem à classe de indivíduos que têm uma relação de tipo **hasSibling**. A ideia-chave é que uma classe de indivíduos é descrita ou definida pelas relações às quais esses indivíduos participam. Em OWL, essas classes podem ser definidas usando restrições (*Restrictions*).

Uma restrição descreve uma classe de indivíduos com base nas relações nas quais os membros da classe participam. Em outras palavras, uma restrição é um tipo de classe, da mesma forma que uma classe nomeada (Named class) (ver item 4.1., p. 19) é também um tipo de classe.

→ Exemplos de restrições

Eis alguns exemplos para ajudar a esclarecer os tipos de classes de indivíduos que podem ser descritas com base em suas propriedades.

- ◆ A classe de indivíduos que possuem pelo menos uma relação de tipo **hasSibling**.
- ◆ A classe de indivíduos que têm pelo menos uma relação de tipo **hasSibling** com membros da classe **Man** — ou seja, coisas que têm pelo menos um irmão que é um homem.
- ◆ A classe de indivíduos que só têm relações de tipo **hasSibling** com indivíduos que são membros da classe **Woman** — ou seja, “coisas” que só têm relações de tipo **hasSibling** e são mulheres (irmãs).
- ◆ A classe de indivíduos que possuem mais de três relações de tipo **hasSibling**.
- ◆ A classe de indivíduos que têm pelo menos uma relação de tipo **hasTopping** com indivíduos que são membros da classe **MozzarellaTopping** — ou seja, a classe de coisas que têm pelo menos um tipo de recheio de muçarela.
- ◆ A classe de indivíduos que só tem uma relação de tipo **temRecheio** (**hasTopping**) com os membros da classe **RecheioDeVerduras** (**VegetableTopping**) — ou seja, a classe de indivíduos que só têm recheios que são recheios de verduras.

Em OWL, todas as classes de indivíduos, acima mencionadas, podem ser descritas usando restrições. As restrições de OWL se dividem em três categorias principais:

- ◆ Restrições de quantificador (*Quantifier Restrictions*).
- ◆ Restrições de cardinalidade (*Cardinality Restrictions*).
- ◆ Restrições de **temValor** (*hasValue Restrictions*).

As restrições de quantificador são compostas por um quantificador, uma propriedade e uma classe nomeada que contém indivíduos os quais atendem a restrição (denominada, *filler*).

Os dois quantificadores disponíveis são:

- ◆ O **quantificador existencial** (ou restrição existencial) (\exists) (*some*): lê-se como pelo menos um, ou algum; em OWL também pode ser lido como **someValuesFrom** (*algunsValoresDe*). As restrições existenciais descrevem classes de indivíduos que participam de pelo menos uma relação por meio de uma propriedade específica com indivíduos que são membros de uma determinada classe. Por exemplo, "a classe de indivíduos que têm **pelo menos uma (some)** relação de tipo **temRecheio** (*hasTopping*) com membros da classe **RecheioDeMuçarela** (*MozzarellaTopping*)". No Protégé 5, a palavra-chave "some" é usada para designar as restrições existenciais.
- ◆ O **quantificador universal** (ou restrição universal) (\forall) (*only*): lê-se como apenas; em OWL também pode ser lido como **allValuesFrom** (*todosValoresDe*). As restrições universais descrevem classes de indivíduos que, para uma determinada propriedade, têm somente relações por meio desta propriedade com indivíduos que são membros de uma determinada classe. Por exemplo, "a classe de indivíduos que têm **somente (only)** relação de tipo **hasTopping** (*temRecheio*) com os membros da classe **VegetableTopping** (*RecheioDeVerduras*)". No Protégé 5, a palavra-chave "only" é usada para designar as restrições universais.

Por exemplo, a restrição **temRecheio some RecheioDeMuçarela** (ou \exists *hasTopping MozzarellaTopping*)²⁰ é constituída pelo quantificador existencial \exists (*some*), pela propriedade **temRecheio** (*hasTopping*), e pelo complemento da restrição (*filler*), a classe **RecheioDeMuçarela** (*MozzarellaTopping*).

Esta restrição descreve o conjunto, ou a classe, de indivíduos que têm, pelo menos, um recheio, e esse recheio é um indivíduo da classe **RecheioDeMuçarela** (*MozzarellaTopping*).

Esta restrição é representada na Figura 4.72 (abaixo): os símbolos em forma de losango (*diamond*) representam indivíduos. Como pode ser visto na Figura 4.29, a restrição descreve uma classe anônima que contém os indivíduos que satisfazem a restrição.

²⁰ Restrição no original em inglês: *hasTopping some MozzarellaTopping* (ou \exists *hasTopping MozzarellaTopping*)

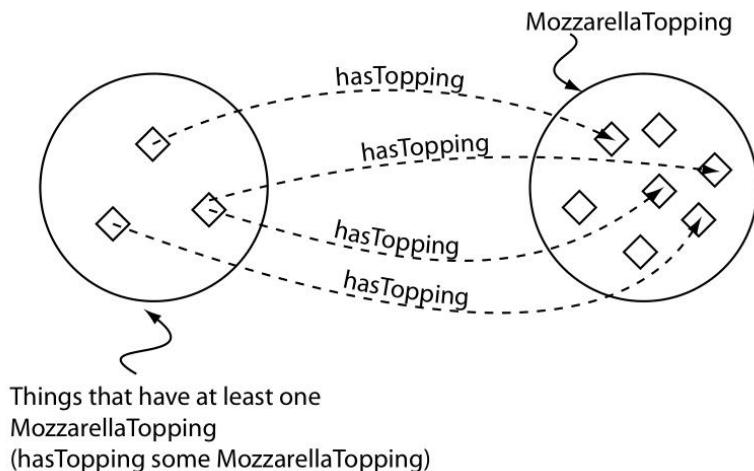


Figura 4.72 – Um exemplo de restrição “some”

A restrição, da figura acima, $\exists \text{ hasTopping } \text{ MozzarellaTopping} \text{ ou hasTopping some MozzarellaTopping}$ descreve a classe de indivíduos que têm, pelo menos, um recheio que é de muçarela (*Mozzarella*).

As restrições de uma classe são exibidas e editadas usando o painel **Classes Description** (Descrição de Classe), mostrando, na figura seguinte, as oito restrições de classes disponíveis no Protégé 5:

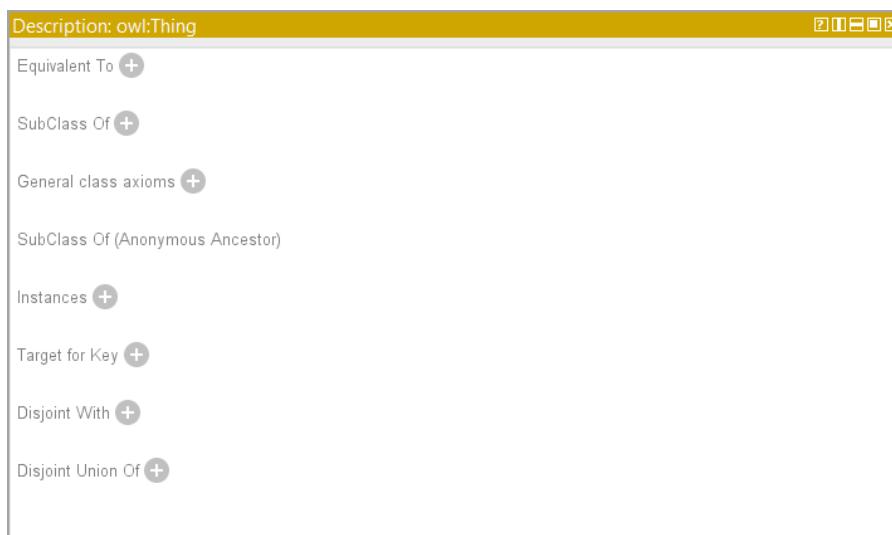


Figura 4.73

O painel **Classes Description** (Descrição de Classes) é o “coração” da guia “Classes”, no Protégé 5, e possui praticamente toda as informações usadas para descrever uma classe. À primeira vista, o painel **Classes Description** (Descrição de Classes) pode parecer complicado, no entanto, ficará evidente que é uma maneira incrivelmente poderosa de descrever e definir classes.

As restrições são usadas em descrições de classes OWL, para especificar (sub)classes anônimas daquelas classes a serem descritas.

4.8.2. Restrições existenciais

As restrições existenciais (\exists) são o tipo mais comum de restrição em ontologias OWL. Para um conjunto de indivíduos, uma restrição existencial (*some*) especifica a existência de uma relação (ou seja, **pelo menos um**) de um desses indivíduos com **outro indivíduo**, o qual é membro de uma classe específica, através da propriedade.

Por exemplo, temMassa **some** MassaDePizza (ou \exists temMassa MassaDePizza)²¹ descreve todos os indivíduos que têm, pelo menos, uma relação com um indivíduo membro da classe **MassaDePizza** (PizzaBase), através da propriedade **temMassa** (hasBase). Em linguagem natural: **todos os indivíduos que têm, pelo menos, uma massa de pizza.**

A restrição existencial também é conhecida como *Some Restrictions* (Algumas Restrições).

💡 Outras ferramentas, artigos e apresentações podem escrever a restrição hasBase some PizzaBase como \exists hasBase PizzaBase. Esse tipo de notação alternativa é conhecido como DL Syntax (sintaxe da Lógica de descrição), que é uma sintaxe mais formal.

Exercício 16: Adicionar uma restrição à classe Pizza que especifica que a classe Pizza deve ter uma MassaDePizza (PizzaBase) [parte 1]

1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **Pizza**.

2º. Clicar no ícone **+ SubClassOf** (SubClasseDe), no painel **Classes Description** (Descrição de classes):

The screenshot shows the Protégé interface with the 'Classes' tab selected. On the left, the 'Class hierarchy' sidebar shows the class 'Pizza' selected. The main workspace displays the 'Description' panel for the class 'Pizza'. The 'SubClass Of' section is active, with a blue highlight around the 'Add' button. Other sections like 'General class axioms', 'Instances', 'Target for Key', 'Disjoint With', and 'Disjoint Union Of' are visible but not highlighted.

Figura 4.74

²¹ Restrição no original em inglês: hasBase **some** PizzaBase (ou \exists hasBase PizzaBase)

3º. Após abertura da caixa de diálogo, selecionar a superclasse `owl:Thing` na aba **Class hierarchy** (Hierarquia de classe) e clicar em **OK**:

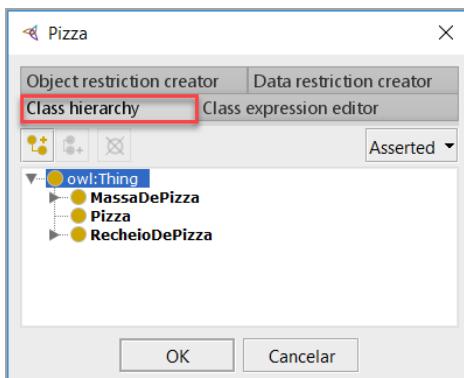


Figura 4.75

⚠ Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:

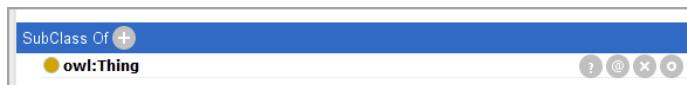


Figura 4.76

O **Class expression editor** (Editor de expressões de classes) permite construir restrições usando nomes de classe, propriedades e indivíduos. É possível arrastar e soltar classes, propriedades e indivíduos na caixa de texto ou digitá-los, a caixa de texto verifica todos os valores digitados e alerta sobre quaisquer tipos de erros.

Para criar uma restrição:

- 1) Digitar o nome da propriedade a ser restrita, que consta na lista de propriedades.
- 2) Digitar um tipo de restrição, que consta na lista de propriedades²². Por exemplo “some” para uma restrição existencial.
- 3) Especificar um complemento da restrição (chamado de *filler*) para a restrição.

Exercício 17: Adicionar uma restrição à classe Pizza que especifica que a classe Pizza deve ter uma MassaDePizza (PizzaBase) [Cont. - Parte2]

1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **Pizza**.

2º. Clicar no ícone **+ SubClassOf** (SubClasseDe), no painel **Classes Description** (Descrição de classes) (Figura 4.74, acima).

3º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição: `temMassa some MassaDePizza`

⚠ (1) propriedade da restrição sobre a classe **Pizza**: “**temMassa**”; (2) o tipo de restrição: “**some**” (\exists) (restrição existencial); (3) o complemento (obrigatório) da restrição (*Filler*): a classe “**MassaDePizza**”.

²² Disponível em: < <http://protegeproject.github.io/protege/class-expression-syntax> >.

4º. Após digitar a restrição completa (com a sintaxe correta, para solicitar ajuda, clicar em **Help**), conforme a figura seguinte, clicar em **OK**:

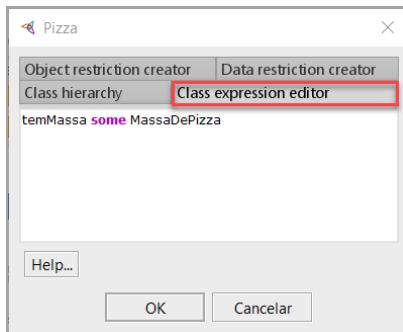


Figura 4.77



Um recurso útil para a construção de expressões é o autocompletar, utilizado para nomes de classes, nomes de propriedades e nomes de indivíduos. O recurso autocompletar é ativado pressionando a tecla [Tab]. No exemplo acima, ao digitar "tem", as opções iniciadas com "tem" são apresentadas, em um menu suspenso. Com as teclas [Up] e [Down] seleciona-se temMassa.



Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:

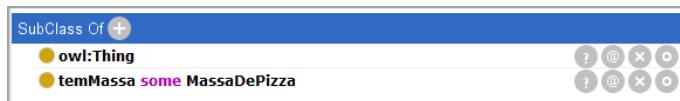


Figura 4.78

Descreveu-se a classe **Pizza** como uma subclasse de **owl:Thing** e como uma subclasse das "coisas" que possuem uma massa, a qual, por sua vez, é algum tipo de **MassaDePizza** (**PizzaBase**). Note-se que tais condições são necessárias, ou seja, se alguma coisa é uma **Pizza** é necessário que tal coisa seja membro da classe **owl:Thing** (em OWL todas as coisas são membros de **owl:Thing**), e ainda é necessário que tal coisa tenha um tipo de **MassaDePizza** (**PizzaBase**).

Em linguagem formal, diz-se que: "Se alguma coisa é uma **Pizza**, é necessário existir uma relação entre tal coisa e um indivíduo membro da classe **MassaDePizza** (**PizzaBase**), através da propriedade **temMassa** (**hasBase**)".

Essa situação é representada na figura seguinte:

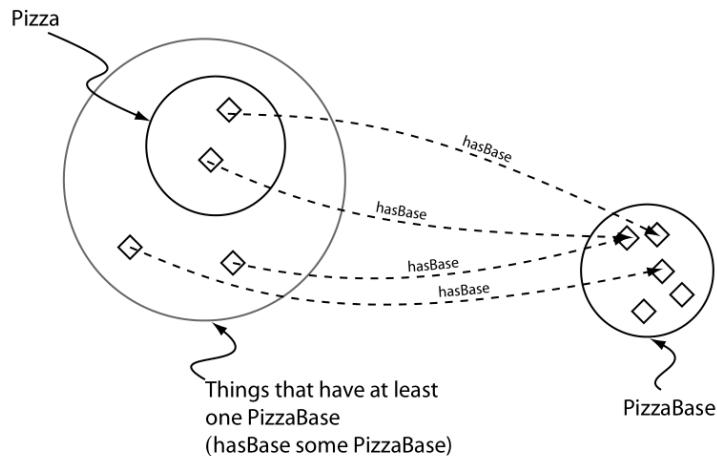


Figura 4.79 - Descrição esquematizada de uma pizza

Assim, para que algo seja uma **Pizza** é necessário que tenha pelo menos uma **MassaDePizza** (PizzaBase); uma **Pizza** é uma subclasse de coisas que têm, pelo menos, um **MassaDePizza** (PizzaBase).

Criação de tipos diferentes de Pizzas:

Deseja-se adicionar diferentes Pizzas à ontologia. Adicionar primeiro a **PizzaMarguerita** (MargheritaPizza), uma **Pizza** que tem recheio de muçarela e tomate. Para manter a ontologia organizada, é preciso agrupar os diferentes tipos de Pizzas na classe **PizzaComNome** (NamedPizza).

Exercício 18: Criar uma subclasse de Pizza chamada PizzaComNome (NamedPizza), e uma subclasse de PizzaComNome (NamedPizza) chamada PizzaMarguerita (MargheritaPizza)

1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **Pizza**.

2º. Criar uma subclasse **PizzaComNome** (NamedPizza) à classe **Pizza**.

3º. Criar uma subclasse **PizzaMarguerita** (MargheritaPizza) à classe **PizzaComNome** (NamedPizza).

⚠ Verificar o resultado gerado, no painel **Class hierarchy (Hierarquia de classe), conforme a figura seguinte:**

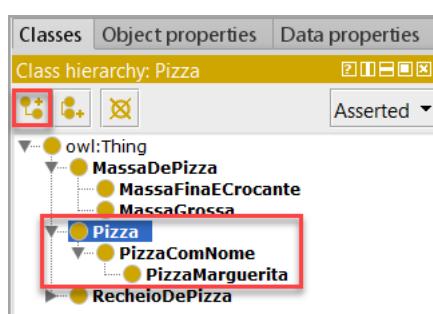


Figura 4.80

- 4º. Inserir um comentário à classe **PizzaMarguerita** (*MargheritaPizza*). Clicar no ícone  **Annotations** (Anotações).

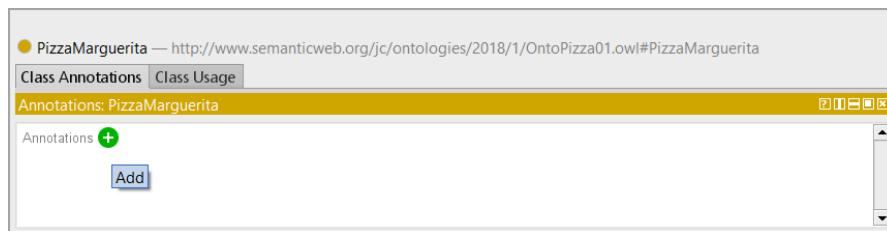


Figura 4.81

- 5º. Após abertura da caixa de diálogo, verificar que a aba **Literal** está aberta e **rdfs:comment** selecionado. Digitar o comentário: *Uma pizza que tem apenas recheio de muçarela e de tomate.*²³

 É possível selecionar o idioma (**Lang**): português.

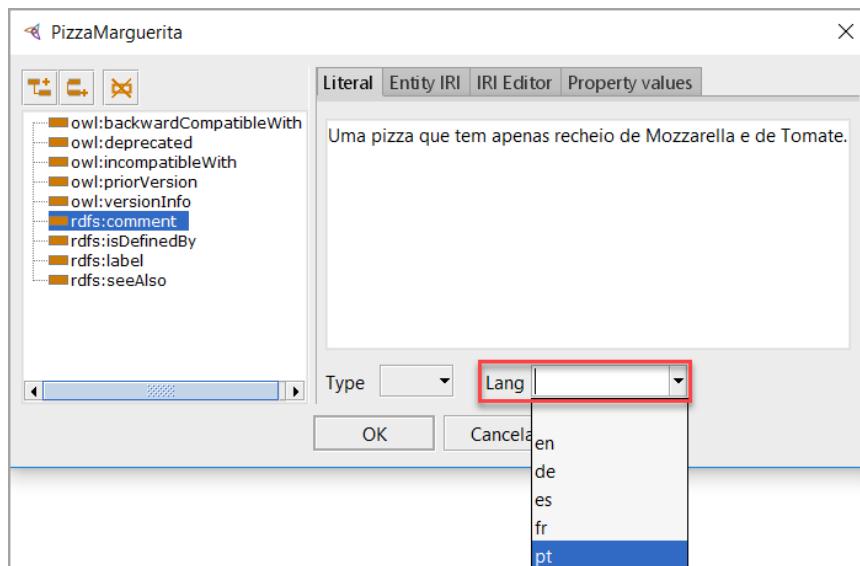


Figura 4.82

 Verificar o resultado gerado, no painel **Annotations**, conforme a figura seguinte:

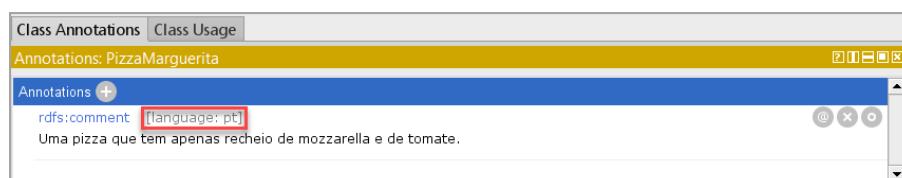


Figura 4.83

 Adicionar comentário é uma boa prática para documentar classes, propriedades, entre outros, durante a edição da ontologia, comunicando o significado pretendido para outros desenvolvedores.

²³ "A pizza that only has Mozzarella and Tomato toppings" (HORRIDGE et al., 2011, p. 44). Tradução de Soares e Almeida (HORRIDGE, 2008).

Tendo criado a classe **PizzaMarguerita** (`MargheritaPizza`) é preciso especificar os recheios dessa **Pizza**. Adicionam-se então duas restrições para dizer que uma **PizzaMarguerita** (`MargheritaPizza`) tem os recheios **RecheioDeMuçarela** (`MozzarellaTopping`) e **RecheioDeTomate** (`TomatoTopping`).

Exercício 19: Criar uma restrição existencial (`some`) para a classe **PizzaMarguerita** (`MargheritaPizza`) que age por meio da propriedade **temRecheio** (`hasTopping`) com o complemento da restrição (`filler`) de **RecheioDeMuçarela** (`MozzarellaTopping`) para especificar que uma **PizzaMarguerita** (`MargheritaPizza`) possui pelo menos um **RecheioDeMuçarela** (`MozzarellaTopping`)

1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaMarguerita** (`MargheritaPizza`).

2º. Clicar no ícone **SubClassOf** (SubClasseDe), no painel **Classes Description** (Descrição de classes):

Figura 4.84

3º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição: `temRecheio some RecheioDeMuçarela`

⚠ (1) propriedade da restrição sobre a classe Pizza: “**temRecheio**”; (2) o tipo de restrição: “**some**” (restrição existencial); (3) o complemento (obrigatório) da restrição (*Filler*): a classe “**RecheioDeMuçarela**”.

4º. Após digitar a restrição completa (com a sintaxe correta, para solicitar ajuda, clicar em **Help**), conforme a figura seguinte, clicar em **OK**:

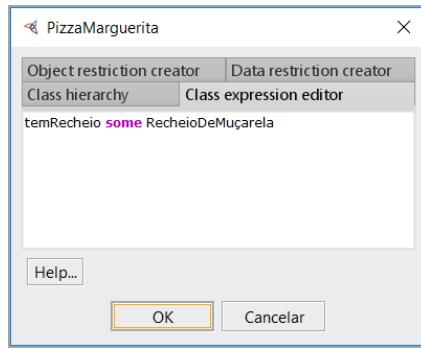


Figura 4.85

⚠ Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:

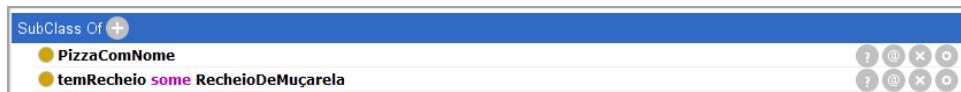


Figura 4.86

Agora deve-se especificar que **PizzaMarguerita** (**MargheritaPizza**) também tem **RecheioDeTomate** (**TomatoTopping**).

Exercício 20: Criar uma restrição existencial (some) para a classe PizzaMarguerita (MargheritaPizza**) que age por meio da propriedade temRecheio (**hasTopping**) com o complemento da restrição (**Filler**) RecheioDeTomate (**TomatoTopping**) para especificar que uma **PizzaMarguerita** (**MargheritaPizza**) **possui pelo menos um RecheioDeTomate** (**TomatoTopping**)**

- 1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaMarguerita** (**MargheritaPizza**).
- 2º. Clicar no ícone **+ SubClassOf** (SubClasseDe), no painel **Classes Description** (Descrição de classes), conforme a Figura 4.84, acima.
- 3º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição: **temRecheio some RecheioDeTomate**
- 4º. Após digitar a restrição completa (com a sintaxe correta). Clicar em **OK**.

⚠ Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:

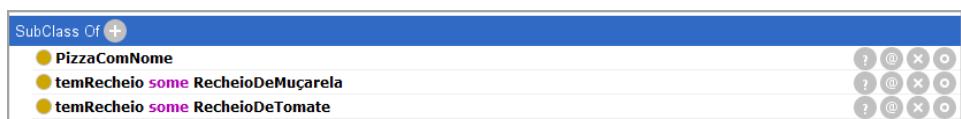


Figura 4.87

Foram adicionadas restrições à **PizzaMarguerita** (**MargheritaPizza**) para informar que uma **PizzaMarguerita** (**MargheritaPizza**) é uma **PizzaComNome** (**NamedPizza**) que tem, **pelo**

menos, um tipo de **RecheioDeMuçarela** (MozzarellaTopping) e, **pelo menos, um** tipo de **RecheioDeTomate** (TomatoTopping).

Formalmente, lê-se: “para que alguma coisa seja membro da classe **PizzaMarguerita** (MargheritaPizza) é necessário que seja membro da classe **PizzaComNome** (NamedPizza); é necessário que seja também um membro da classe anônima de coisas que estão ligadas a pelo menos um membro da classe **RecheioDeMuçarela** (MozzarellaTopping) via a propriedade **temRecheio** (**hasTopping**), e ainda é necessário que seja um membro da classe **RecheioDeTomate** (TomatoTopping) via propriedade **temRecheio** (**hasTopping**)”.

Pode-se criar uma classe para representar a **PizzaAmericana** (AmericanPizza), que tem **RecheioDeCalabresa** (PepperoniTopping), **RecheioDeMuçarela** (MozzarellaTopping) e **RecheioDeTomate** (TomatoTopping).

A classe **PizzaAmericana** (AmericanPizza) é similar à classe **PizzaMarguerita** (MargheritaPizza): uma **PizzaAmericana** (AmericanPizza) é quase uma **PizzaMarguerita** (MargheritaPizza) com a diferença de que tem um recheio a mais (Calabresa).

Dessa forma, pode ser realizado uma **clonagem** da classe **PizzaMarguerita** (MargheritaPizza) e então adicionar uma restrição extra para informar que ela tem um **RecheioDeCalabresa** (PepperoniTopping).

Exercício 21: Criar a classe PizzaAmericana (AmericanPizza), clonando a classe PizzaMarguerita (MargheritaPizza) e modificando sua descrição, acrescentando um recheio: RecheioDeCalabresa (PepperoniTopping)

1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaMarguerita** (MargheritaPizza).

2º. Abrir a aba **Edit** e selecionar a função **Duplicate selected class...** (Duplicar a classe selecionada)

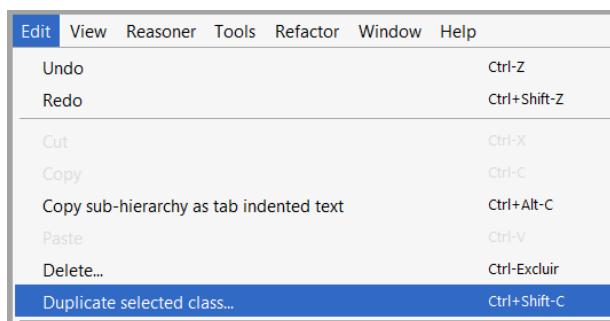


Figura 4.88

3º. Após abertura da caixa de diálogo, aparece uma mensagem de erro (em vermelho, na Figura 4.89, abaixo). A substituição do nome da classe **PizzaMarguerita**, pela classe **PizzaAmericana**, vai gerar um clone de **PizzaMarguerita** com as mesmas características (restrições, etc). Clicar em **OK**.

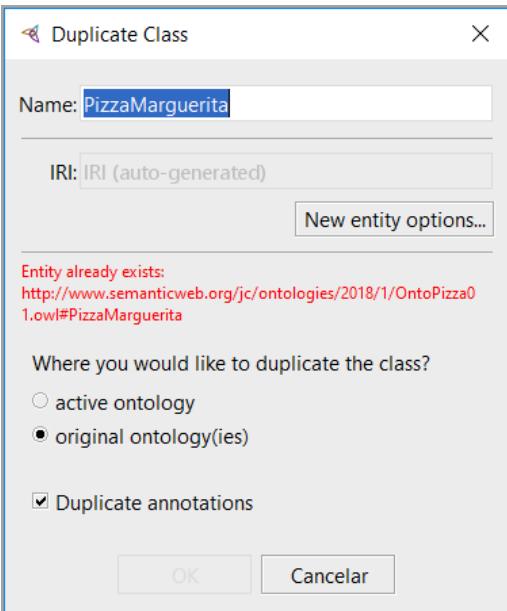


Figura 4.89

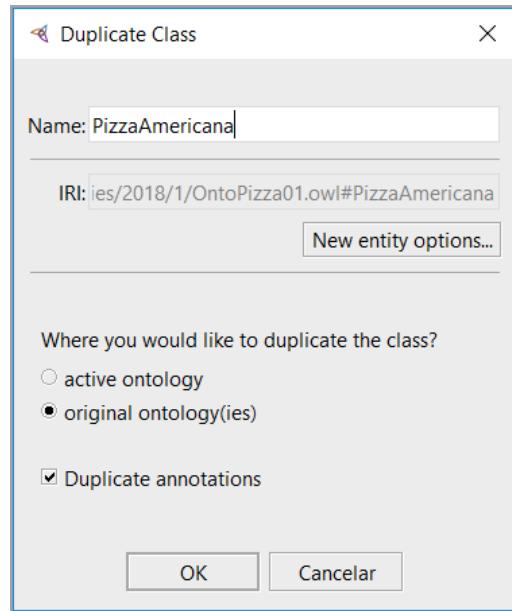


Figura 4.90

- 4º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaAmericana** (AmericanPizza).
- 5º. Clicar no ícone **SubClassOf** (SubClasseDe), no painel **Classes Description** (Descrição de classes), conforme a Figura 4.84, acima.
- 6º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição: temRecheio **some** RecheioDeCalabresa
- 7º. Após digitar a restrição completa (com a sintaxe correta). Clicar em **OK**.

Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:

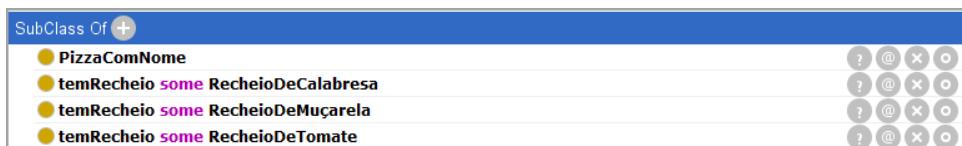


Figura 4.91

Exercício 22: Criar as classes **PizzaAmericanaPicante** (AmericanHotPizza) e **PizzaSoho** (SohoPizza), por clonagem

- 1º. **[Clone n. 1]** Uma **PizzaAmericanaPicante** (AmericanHot Pizza) é quase uma **PizzaAmericana** (AmericanPizza), mas possui uma **RecheioDePimentaMexicana** (JalapenoPepperTopping). Criar uma classe clonada da classe **PizzaAmericana** (AmericanPizza), adicionando uma restrição existencial (some) à propriedade **temRecheio** (hasTopping) com um complemento da restrição (Filler) **RecheioDePimentaMexicana** (JalapenoPepperTopping).

- 2º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaAmericana** (AmericanPizza).
- 3º. Abrir a aba **Edit** e selecionar a função **Duplicate selected class...** (Duplicar a classe selecionada) para criar o clone **PizzaAmericanaPicante** (AmericanHot Pizza).
- 4º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaAmericanaPicante** (AmericanHot Pizza).
- 5º. Clicar no ícone  **SubClassOf** (SubClasseDe), no painel **Classes Description** (Descrição de classes).
- 6º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição: `temRecheio some RecheioDePimentaMexicana`
- 7º. Após digitar a restrição completa (com a sintaxe correta). Clicar em **OK**.

 Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:



Figura 4.92

- 8º. **[Clone n. 2]** Uma **PizzaSoho** (SohoPizza) é quase uma **PizzaMarguerita** (MargheritaPizza), mas tem recheios adicionais de **RecheioDeAzeitona** (OliveTopping) e **RecheioDeParmesão** (ParmesanTopping). Criar uma classe clonada da classe **PizzaMarguerita** (MargheritaPizza) e adicionando duas restrições existenciais (some) à propriedade **temRecheio** (hasTopping) com dois complementos da restrição (*Filler*): **RecheioDeAzeitona** (OliveTopping) e **RecheioDeParmesão** (ParmesanTopping).
- 9º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaMarguerita** (MargheritaPizza).
- 10º. Abrir a aba **Edit** e selecionar a função **Duplicate selected class...** (Duplicar a classe selecionada) para criar o clone **PizzaSoho** (SohoPizza).
- 11º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaSoho** (SohoPizza).
- 12º. Clicar no ícone  **SubClassOf** (SubClasseDe), no painel **Classes Description** (Descrição de classes).
- 13º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição: `temRecheio some RecheioDeAzeitona`
- 14º. Após digitar a restrição completa (com a sintaxe correta). Clicar em **OK**.

- 15º. Clicar no ícone  **SubClassOf** (SubClasseDe), no painel **Classes Description** (Descrição de classes).
- 16º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição: temRecheio **some RecheioDeParmesão**
- 17º. Após digitar a restrição completa (com a sintaxe correta). Clicar em **OK**.

 Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:

SubClass Of	?	@	X	O
PizzaComNome	?	@	X	O
temRecheio some RecheioDeAzeitona	?	@	X	O
temRecheio some RecheioDeMuçarela	?	@	X	O
temRecheio some RecheioDeParmesão	?	@	X	O
temRecheio some RecheioDeTomate	?	@	X	O

Figura 4.93

Após criar essas pizzas é preciso torná-las disjuntas.

Exercício 23: Tornar disjuntas todas as subclasses da classe PizzaComNome (NamedPizza)

- 1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaMarguerita (MargheritaPizza)** ou qualquer outra Pizza do mesmo nível (irmãs).
- 2º. Abrir a aba **Edit** e selecionar a função **Make all primitive siblings disjoint** (Tornar disjuntas todas as primitivas irmãs):

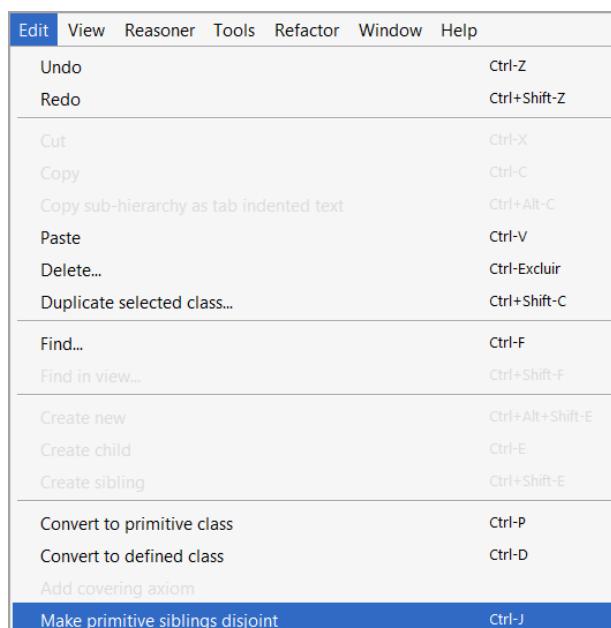


Figura 4.94

 Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:

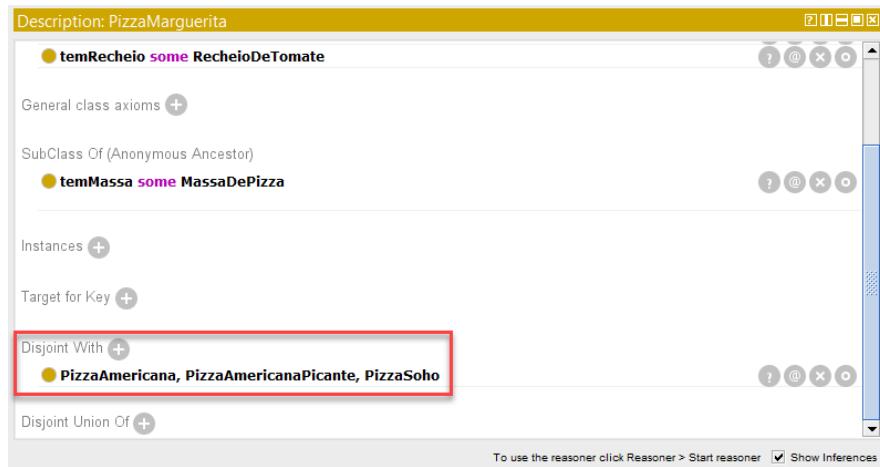


Figura 4.95

4.9. Uso de um mecanismo de inferência (*Reasoner*)

Uma característica importante de ontologias descritas com o OWL-DL é a possibilidade de processamento por um mecanismo de inferência (*Reasoner*). Um relevante serviço oferecido por mecanismos de inferência (MI) é o teste para saber se uma classe é, ou não é, uma subclasse de outra classe (*subsumption test*), ou seja, as descrições das classes (condições) são utilizadas para determinar se existe entre elas uma relação superclasse/subclasse. Através de tais testes em todas as classes de uma ontologia é possível inferir automaticamente a hierarquia de classes da ontologia.

Outro serviço padrão oferecido pelo mecanismo de inferência é o de *consistency checking* (verificação da consistência). Baseado na descrição (condições) de uma classe, o mecanismo de inferência pode verificar se é possível, ou não, que uma classe possua instâncias. Uma classe é considerada inconsistente se não é possível a ela ter instâncias.

Os mecanismos de inferência (*Reasoner*) às vezes são chamados classificadores. A classificação, no entanto, não é o único serviço de inferência fornecido pelos mecanismos de inferência. Por exemplo, um MI também executa a verificação de consistência (*consistency checking*) que é diferente da classificação. Portanto, o termo adequado é “mecanismo de inferência” (*Reasoner*).

4.9.1. Conhecendo o mecanismo de inferência (MI)

O Protégé 5 permite que diferentes mecanismos de inferência OWL sejam conectados (vários aplicativos disponíveis), o mecanismo de inferência (MI) fornecido com Protégé é chamado **HermiT**.

No Protégé 5, a hierarquia de classe construída manualmente é chamada de *asserted hierarchy* (hierarquia declarada); a hierarquia de classes calculada automaticamente pelo MI é chamada *Inferred hierarchy* (hierarquia inferida).

Para lançar o processo de inferência, ou seja, classificar automaticamente a ontologia e verificar sua consistência, é preciso abrir a aba **Reasoner** (Mecanismo de inferência – MI), e clicar em **Start Reasoner** (Figura 4.97, p. 64).

Quando a hierarquia inferida (*Inferred hierarchy*) for calculada, o resultado é apresentado no painel **Classe hierarchy** e a guia **Inferred** selecionada (Figura 4.100, p. 65). Se uma classe inconsistente foi encontrada, ela aparece em vermelho.

A tarefa de calcular a hierarquia de classe inferida é também conhecida como *classificação da ontologia*.

4.9.2. Classes inconsistentes

Para demonstrar o uso do MI na detecção de inconsistências foi criada uma classe, sendo subclasse de **RecheioDeQueijo** (CheeseTopping) e de **RecheioDeVerduras** (VegetableTopping). Esta estratégia é normalmente usada para verificar se a ontologia foi construída corretamente. As classes adicionadas para testar a integridade da ontologia são conhecidas como **ProbeClasses** (Classes de Investigação).

Exercício 24: Adicionar uma classe de investigação (Probe Class) denominada InvestigaçãoinconsistenteDoRecheio (ProbeInconsistentTopping) como subclasse de RecheioDeQueijo (CheeseTopping) e de RecheioDeVerduras (VegetableTopping)

- 1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **RecheioDeQueijo** (CheeseTopping).
- 2º. Criar uma subclasse  **InvestigaçãoinconsistenteDoRecheio** (ProbeInconsistentTopping) à classe **RecheioDeQueijo** (CheeseTopping) (Figura 4.96, abaixo ①)
- 3º. Inserir um comentário à classe **InvestigaçãoinconsistenteDoRecheio** (ProbeInconsistentTopping). Isso possibilita que outra pessoa, ao analisar a ontologia, entenda que a classe foi deliberadamente criada como inconsistente.
- 4º. No painel **Annotations**, clicar no ícone  para inserir comentários.
- 5º. A caixa de diálogo **Create Annotations** aparece. Verificar que a aba **Literal** está aberta e `rdfs:comment` selecionado. Digitar o comentário: **Essa classe será marcada como inconsistente quando a ontologia for classificada.** Clicar em: **OK** (Figura 4.96, abaixo ②)
- 6º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **InvestigaçãoinconsistenteDoRecheio** (ProbeInconsistentTopping).
- 7º. Clicar no ícone  **SubClassOf** (SubClasseDe), no painel **Classes Description** (Descrição de classes). Após abertura da caixa de diálogo, selecionar a superclasse

RecheioDeVerduras (`VegetableTopping`), na aba **Class hierarchy** (Hierarquia de classe) e clicar em **OK** (Figura 4.96, abaixo ③)

⚠ Verificar os resultados gerados, nos três painéis, conforme a figura seguinte:

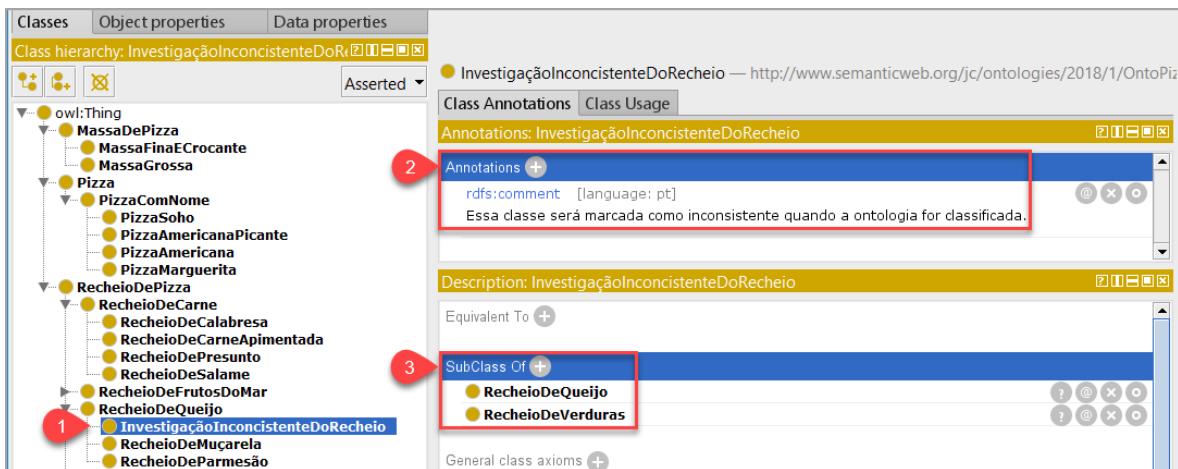


Figura 4.96

Ao examinar a hierarquia, a classe **InvestigaçāoInconsistenteDoRecheio** (`ProbeInconsistentTopping`) aparece como subclasse de **RecheioDeQueijo** (`CheeseTopping`) e subclasse de **RecheioDeVerduras** (`VegetableTopping`). Isso quer dizer que **InvestigaçāoInconsistenteDoRecheio** (`ProbeInconsistentTopping`) é um **RecheioDeQueijo** (`CheeseTopping`) e um **RecheioDeVerduras** (`VegetableTopping`), ou seja, todos os indivíduos que são membros de **InvestigaçāoInconsistenteDoRecheio** (`ProbeInconsistentTopping`) são também (necessariamente) membros da classe **RecheioDeQueijo** (`CheeseTopping`) e (necessariamente) membros de **RecheioDeVerduras** (`VegetableTopping`). Isto é incorreto visto que algo não pode ser queijo e vegetal ao mesmo tempo.

Exercício 25: Verificar a inconsistência da classe **InvestigaçāoInconsistenteDoRecheio** (`ProbeInconsistentTopping`)

1º. Abrir a aba **Reasoner** (Mecanismo de inferência – MI), para lançar o processo de inferência, clicar em: **Start Reasoner**

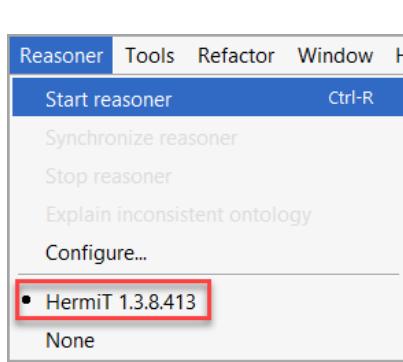


Figura 4.97



Figura 4.98

💡 O indicador de estatuto do MI indica que ele está ativo (parte inferior da tela do Protégé, lado direito).

 O Mecanismo de inferência usado é Hermit.

 Verificar os resultados gerados, nos dois painéis de hierarquia **declarada** (*Asserted hierarchy*) e de **hierarquia inferida** (*Inferred hierarchy*), conforme as figuras seguintes:

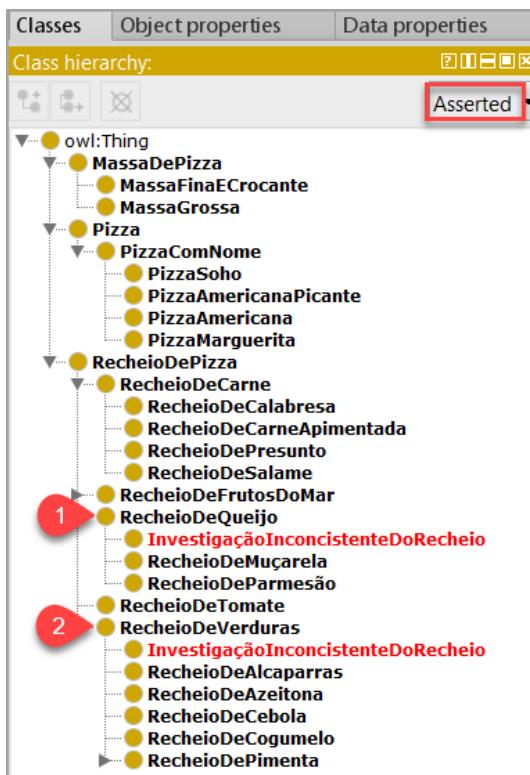


Figura 4.99

 O painel de **Class hierarchy** (Hierarquia de classe) oferece duas visualizações da hierarquia de classe. A figura acima apresenta a **hierarquia declarada** (Asserted hierarchy). A classe **InvestigaçāoInconsistenteDoRecheio** (ProbeInconsistentTopping) foi declarada pelo MI inconsistente (vermelho), nas duas classes ① e ②.

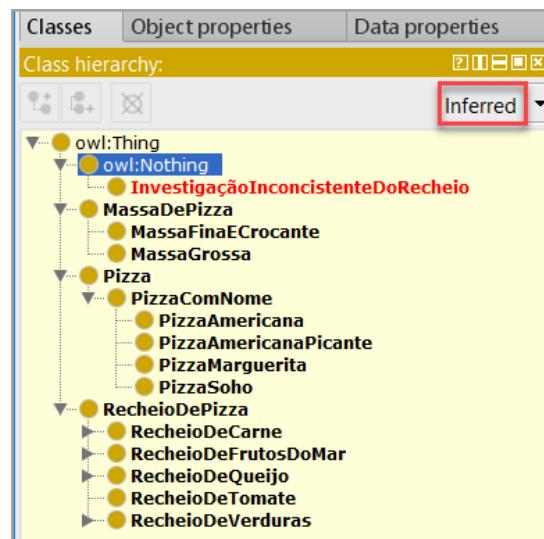


Figura 4.100

 A figura acima apresenta a **hierarquia inferida** (Inferred hierarchy). A classe **InvestigaçāoInconsistenteDoRecheio** (ProbeInconsistentTopping) foi declarada pelo MI inconsistente (vermelho).

 Esse painel é visível apenas quando o MI está ativado (fundo amarelo). Para desativar o MI: clicar em **Stop Reasoner**, na aba **Reasoner**:

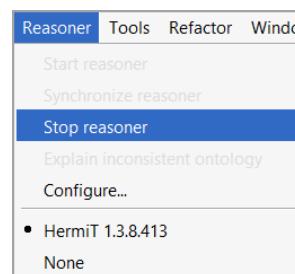


Figura 4.101

Análise da situação: sabe-se que uma coisa não pode ser queijo e vegetal ao mesmo tempo. Uma coisa não pode ser instância de **RecheioDeQueijo** (CheeseTopping) e instância

de **RecheioDeVerduras** (VegetableTopping) simultaneamente. Entretanto, o nome para as classes é escolhido pelo desenvolvedor.

Para um MI, os nomes não têm significados, e ele não pode determinar se alguma coisa é inconsistente baseando-se em nomes. O motivo de a classe **InvestigaçãoinconsistenteDoRecheio** (ProbeInconsistentTopping) ser detectada como inconsistente é que suas superclasses **RecheioDeVerduras** (VegetableTopping) e **RecheioDeQueijo** (CheeseTopping) são disjuntas.

Assim, indivíduos que são membros de **RecheioDeQueijo** (CheeseTopping) não podem ser membros de **RecheioDeVerduras** (VegetableTopping) e vice-versa.

Exercício 26: Remover a declaração disjunta entre as classes RecheioDeQueijo (CheeseTopping) e RecheioDeVerduras (VegetableTopping) para observar o resultado

- 1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **RecheioDeQueijo** (CheeseTopping).
- 2º. O campo **Disjoint with** (Disjunta com) deve conter as classes irmãs do **RecheioDeQueijo** (CheeseTopping): **RecheioDeVerduras** (VegetableTopping), **RecheioDeFrutosDoMar** (SeafoodTopping) e **RecheioDeCarne** (MeatTopping), conforme a figura seguinte:

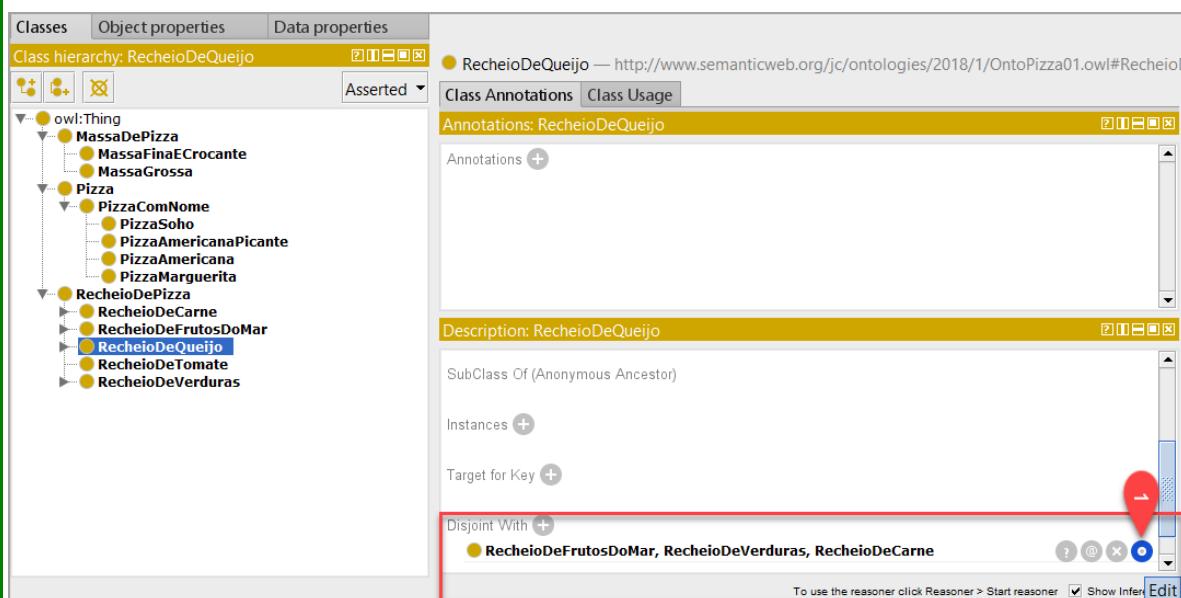


Figura 4.102

- 3º. Abrir o editor **Expression Editor** (Editor de expressão), clicando no ícone (**Edit**)
 1. Após abertura da caixa de diálogo, selecionar **Expression Editor** (Editor de expressão) e deletar apenas a classe **RecheioDeVerduras** (VegetableTopping) (quadro verde, na figura seguinte), para remover o axioma de disjunção que mantinha as classes **RecheioDeQueijo** (CheeseTopping) e **RecheioDeVerduras** (VegetableTopping) disjuntas.

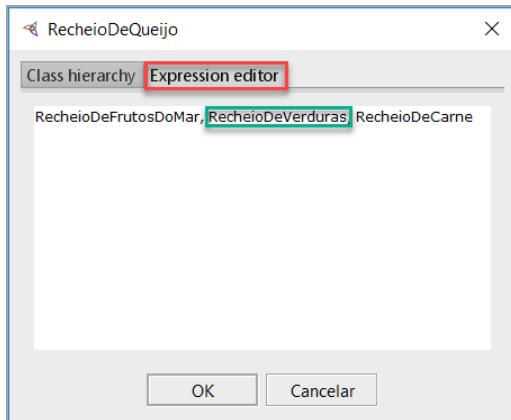


Figura 4.103

- 4º. Abrir a aba **Reasoner** (Mecanismo de inferência – MI), para lançar o processo de inferência, clicar em: **Start Reasoner**. O MI não apresenta mais inconsistências (em vermelho) nos dois painéis: de hierarquia declarada (*Asserted hierarchy*) e de hierarquia inferida (*Inferred hierarchy*).

Observe-se que **InvestigaõInconsistenteDoRecheio** (ProbeInconsistentTopping) não é mais inconsistente. Isto significa que indivíduos que são membros da classe **InvestigaõInconsistenteDoRecheio** (ProbeInconsistentTopping) são também membros da classe **RecheioDeQueijo** (CheeseTopping) e de **RecheioDeVerduras** (VegetableTopping). Ou seja, estabeleceu-se (de forma errônea) que alguma coisa pode ser um queijo e um vegetal ao mesmo tempo.

Isto ilustra a importância do cuidado no uso de axiomas de disjunção. **As classes OWL se sobrepõem até que sejam declaradas disjuntas.** Se algumas classes não são disjuntas, resultados inesperados podem surgir. Se certas classes são marcadas como disjuntas de forma incorreta, novamente resultados inesperados podem surgir.

Exercício 27: Corrigir a ontologia, tornando disjuntas as classes **RecheioDeQueijo (CheeseTopping) e **RecheioDeVerduras** (VegetableTopping)**

- 1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **RecheioDeQueijo** (CheeseTopping).
- 2º. O campo **Disjoint with** (Tornar disjunto de) deve conter as classes irmãs do **RecheioDeQueijo** (CheeseTopping): **RecheioDeFrutosDoMar** (SeafoodTopping) e **RecheioDeCarne** (MeatTopping), conforme a figura seguinte:

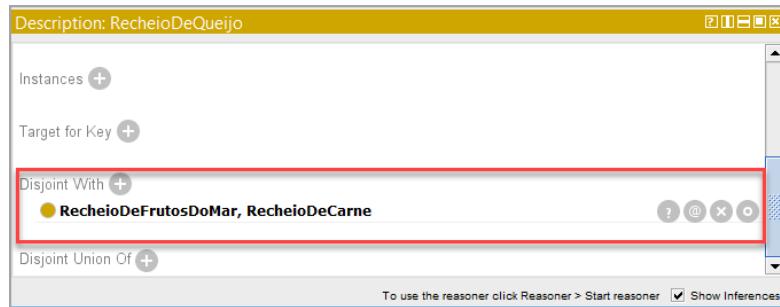


Figura 4.104

- 2º. Clicar no ícone **Disjoint With** (Tornar disjunto de), no painel **Description** (Descrição). Selecionar a classe **RecheioDeVerduras** (VegetableTopping) e Clicar em: **OK**.

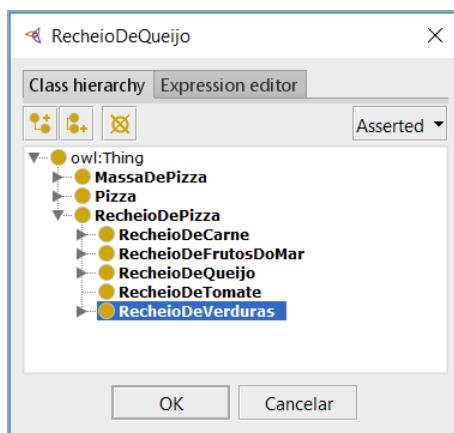


Figura 4.105

A classe **RecheioDeQueijo** (**CheeseTopping**) deve ser novamente disjunta de **RecheioDeVerduras** (**VegetableTopping**):



Figura 4.106

- 3º. Verificar se o axioma de disjunção foi adicionado corretamente. Abrir a aba **Reasoner** (Mecanismo de inferência – MI), para ativar o processo de inferência, clicar em: **Start Reasoner**. A ontologia é classificada e a classe **InvestigaçãoInconsistenteDoRecheio** (**ProbeInconsistentTopping**) é, novamente, marcada em vermelho, indicando inconsistência, conforme apresentado na Figura 4.99 e na Figura 4.100 (p.65).

4.10. Condições necessárias e suficientes (classes primitivas e definidas)

As classes criadas até agora foram descritas apenas com condições necessárias. A condição necessária pode ser lida da seguinte forma: se alguma coisa é um membro da classe então

é necessário que preencha essas condições. O uso de condições necessárias não permite dizer: se alguma coisa preenche essas condições então deve ser um membro dessa classe.

Se uma classe tem apenas condições necessárias, ela é conhecida como uma *classe primitiva*.

Por exemplo, uma subclasse de **Pizza** chamada **PizzaDeQueijo** (**CheesyPizza**) é uma **Pizza** com, pelo menos, um tipo de **RecheioDeQueijo** (**CheeseTopping**).

Exercício 28: Criar uma subclasse de Pizza chamada PizzaDeQueijo (CheesyPizza), com pelo menos um recheio que é um tipo de RecheioDeQueijo (CheeseTopping)

1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **Pizza**.

2º. Criar uma subclasse  **PizzaDeQueijo** (**CheesyPizza**) à classe **Pizza**.

3º. Selecionar a nova classe **PizzaDeQueijo** (**CheesyPizza**) e clicar no ícone  **SubClassOf** (**SubClasseDe**), no painel **Classes Description** (Descrição de classes).

4º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição: `temRecheio some RecheioDeQueijo`

 (1) propriedade da restrição sobre a classe **Pizza**: “**temRecheio**”; (2) o tipo de restrição: “**some**” (restrição existencial); (3) o complemento (obrigatório) da restrição (*Filler*): a classe “**RecheioDeQueijo**”.

5º. Após digitar a restrição completa (com a sintaxe correta), conforme a figura seguinte, clicar em **OK**:

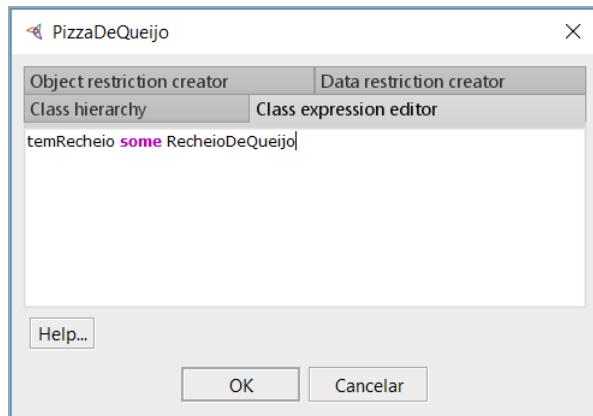


Figura 4.107

 Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:

Figura 4.108

A atual descrição de **PizzaDeQueijo** (CheesyPizza) indica que se alguma coisa é membro da classe **PizzaDeQueijo** (CheesyPizza) é necessário que seja membro da classe **Pizza** e, além disso, ainda é necessário que tenha pelo menos um recheio que seja membro da classe **RecheioDeQueijo** (CheeseTopping). Para dizer isto, usam-se **condições necessárias** (**SubClassOf**).

Considere-se agora um indivíduo qualquer. Sabe-se que esse indivíduo é membro da classe **Pizza**; sabe-se também que tem, pelo menos, um tipo de **RecheioDeQueijo** (CheeseTopping).

Entretanto, pela atual descrição de **PizzaDeQueijo** (CheesyPizza), essas informações **não são suficientes** para determinar que o indivíduo é membro da classe **PizzaDeQueijo** (CheesyPizza).

Para que isto seja possível é preciso mudar as condições de **PizzaDeQueijo** (CheesyPizza), de condições necessárias para **condições necessárias e suficientes**. As condições não vão ser apenas necessárias para associação à classe **PizzaDeQueijo** (CheesyPizza), elas vão ser também suficientes para determinar que qualquer indivíduo que as satisfaça é um membro de **PizzaDeQueijo** (CheesyPizza).

Uma classe que tem, pelo menos, um conjunto de *condições necessárias e suficientes* é conhecida como *Defined Class* (*Classe Definida*).

As *condições necessárias* são chamadas *Subclasses* (no Protégé 5). As *condições necessárias e suficientes* são conhecidas como *Equivalent classes* (*Classes equivalentes*).

Para converter as *condições necessárias* em *condições necessárias e suficientes*, as condições devem ser movidas do campo **Subclass Of** (no painel de descrição da classe) (Figura 4.109) para o campo **Equivalent To** (Figura 4.111). Isso pode ser feito com a opção **Convert to defined class** (Converter em uma classe definida), na aba **Edit** (Figura 4.110).

Exercício 29: Converter as condições necessárias da classe **PizzaDeQueijo** (CheesyPizza) em condições necessárias e suficientes

1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaDeQueijo** (CheesyPizza).

The screenshot shows the Protégé interface with the following details:

- Left Panel (Classes):** Displays the class hierarchy under "owl:Thing". Key nodes include **Pizza**, **PizzaDeQueijo**, **MassaDePizza**, **MassaFinaECrocante**, **MassaGrossa**, **RecheioDePizza**, **RecheioDeCarne**, **RecheioDeFrutosDoMar**, **RecheioDeQueijo**, **InvestigaçãoInconscienteDoRecheio**, **RecheioDeMuçarela**, **RecheioDeParmesão**, **RecheioDeTomate**, **RecheioDeVerduras**, **InvestigaçãoInconscienteDoRecheio**, **RecheioDeAlcaparras**, **RecheioDeAzeitona**, **RecheioDeCebola**, **RecheioDeCogumelo**, and **RecheioDePimenta**.
- Top Tab Bar:** Shows "Classes", "Object properties", and "Data properties".
- Annotations Tab:** Shows annotations for **PizzaDeQueijo**. A red box highlights the "SubClass Of" section, which contains the axiom **temRecheio some RecheioDeQueijo**.
- Bottom Status Bar:** Includes links to "Reasoner" and "Show Inferences".

Figura 4.109

2º. Abrir a aba **Edit** e selecionar a função **Convert to defined class** (Converter em uma classe definida):

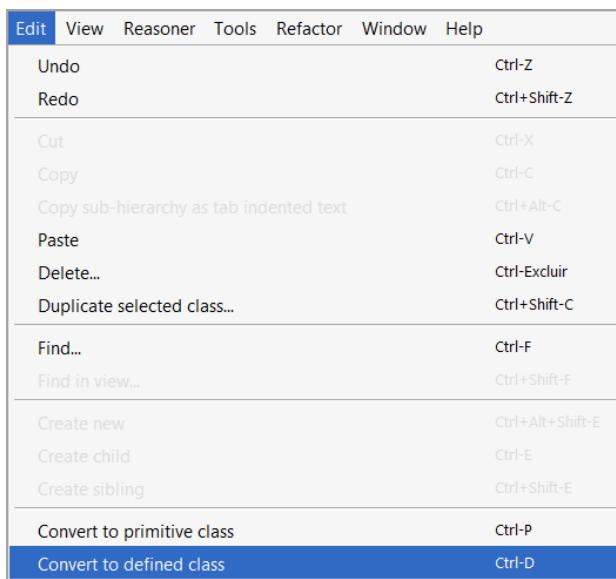


Figura 4.110

3º. A classe **PizzaDeQueijo** (CheesyPizza) convertida:

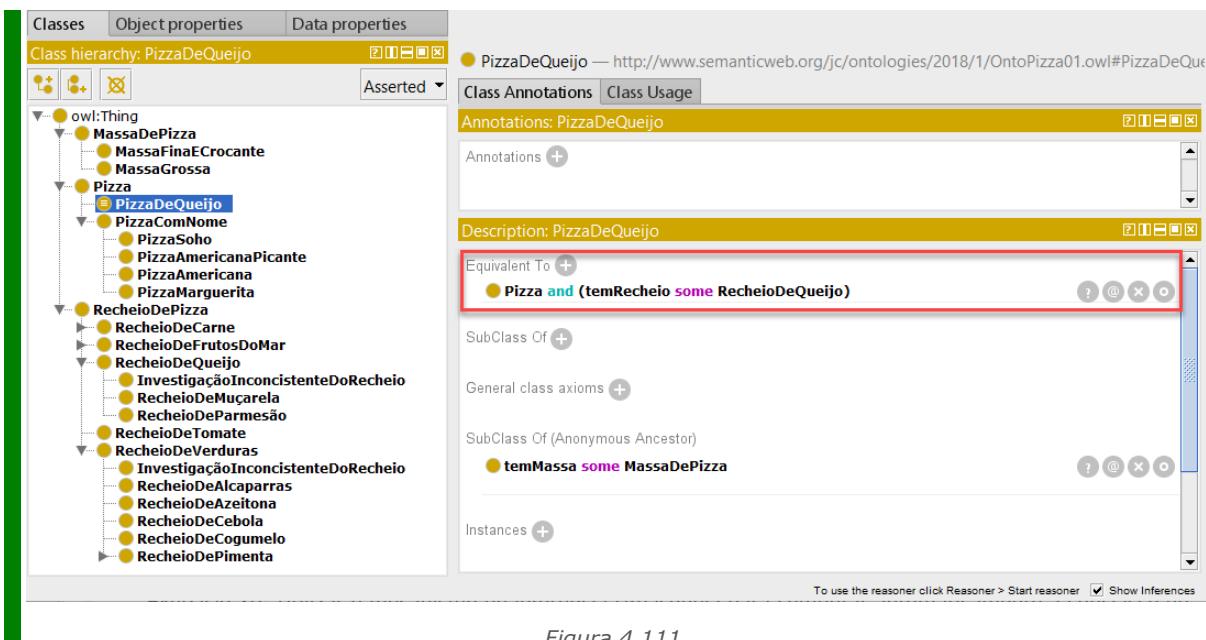


Figura 4.111

A descrição de **PizzaDeQueijo** (CheesyPizza) foi convertida em uma definição. Se alguma coisa é uma **PizzaDeQueijo** (CheesyPizza) é necessário que seja uma **Pizza**, também é necessário que **pelo menos um** de seus recheios seja membro da classe **RecheioDeQueijo** (CheeseTopping).

Além disso, se um indivíduo é um membro da classe **Pizza** e ele tem, pelo menos, um recheio que é membro da classe **RecheioDeQueijo** (CheeseTopping) então essas condições são suficientes para determinar que o indivíduo deve ser um membro da classe **PizzaDeQueijo** (CheesyPizza).

A noção de condições necessárias e suficientes é ilustrada na Figura 4.112 e na Figura 4.113.

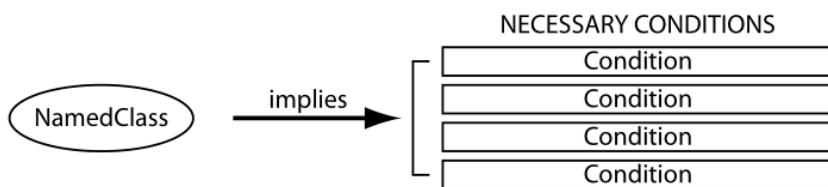


Figura 4.112 – Condições necessárias

Na Figura 4.112, se um indivíduo é membro de **NamedClass** (ClasseNomeada) então é obrigatório que satisfaça as condições. Entretanto, se algum indivíduo satisfaz as condições necessárias, não se pode dizer que seja membro de **NamedClass** (ClasseNomeada): as condições não são suficientes para que se possa dizer isso. Tal fato é indicado pela direção da seta.

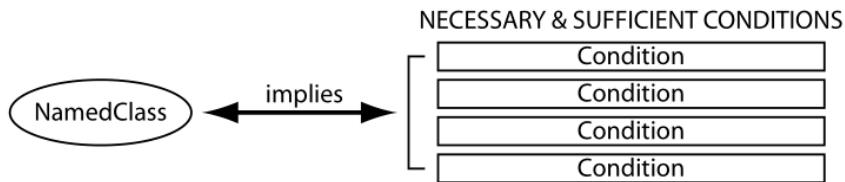


Figura 4.113 - Condições necessárias e suficientes

Na Figura 4.113, se um indivíduo é membro de **NamedClass** (ClasseNomeada) então é obrigatório que satisfaça as condições. Se algum indivíduo satisfaz as condições então é obrigatório que seja membro de **NamedClass** (ClasseNomeada). Tal fato é indicado pela seta bidirecional.

Resumindo... se a classe A é descrita por condições necessárias, então pode-se dizer que se um indivíduo é membro de A, ele deve satisfazer as condições. Não se pode dizer que qualquer indivíduo (aleatório) que satisfaça tais condições é um membro da classe A.

Entretanto, se a classe A é definida usando condições necessárias e suficientes pode-se dizer que se um indivíduo é membro da classe A ele deve satisfazer as condições, e pode-se dizer que se qualquer indivíduo satisfaz essas condições então ele deve ser membro de A. As condições não são apenas necessárias para a associação com A, mas também suficientes de forma a determinar que, se alguma coisa satisfaz essas condições, é um membro de A.

Como isso é útil na prática?

Suponha-se outra classe B e que qualquer indivíduo que seja membro da classe B também satisfaça as condições que definem a classe A. Pode-se determinar que a classe B é subclasse de A. Verifica-se que a relação subclasse/superclasse de classes (*subsumption relationship*) é uma tarefa básica de um MI de lógica descritiva, e que é possível usá-lo para computar automaticamente a hierarquia.

No Protégé 5, as condições necessárias e suficientes são definidas na seção **Equivalent To** (Defined Classes) e as condições necessárias, na seção **SubClass Of** (Primitives Classes), disponíveis em cada uma das grandes seções (guias) do Protégé 5: Classes, Object Properties, DataType Properties, Individuals.

⚠️ Em OWL é possível ter conjuntos múltiplos de condições necessárias e suficientes (ver seção 7.5. , p. 125).

→ **Classes Primitivas (Primitives Classes) e classes definidas (Defined Classes):**

As classes que têm, pelo menos, um conjunto de condições necessárias e suficientes são conhecidas como Defined Classes (Classes Definidas): tais classes têm uma definição, e qualquer indivíduo que satisfaça tal definição pertence à classe.

As classes que não tem nenhum conjunto de condições necessárias e suficientes (apenas condições necessárias) são conhecidas como Primitive Classes (Classes Primitivas).

⚠️ No Protégé 5, as Defined Classes (Classes Definidas) possuem um ícone com fundo alaranjado, com 3 linhas . As Primitive Classes (Classes Primitivas) possuem um ícone com fundo alaranjado .

⚠ É importante observar que o MI pode classificar automaticamente apenas as classes definidas, ou seja, classes com pelo menos um conjunto de *condições necessárias e suficientes*.

4.11. Classificação automática

Um dos benefícios em construir uma ontologia com a sublinguagem OWL-DL é a possibilidade de computar automaticamente a hierarquia de classes através de um MI. No caso de ontologias muito grandes (milhares de classes), esse auxílio automático para computar relacionamentos subclasse-superclasse é vital. Sem um MI é muito difícil manter grandes ontologias em um estado logicamente correto.

Em casos em que as ontologias têm classes com muitas subclasses (herança múltipla) é uma boa prática construir uma hierarquia de classes como uma árvore simples. Dessa forma, as classes na Asserted Hierarchy (Hierarquia Declarada, aquela construída manualmente) não têm mais do que uma subclasse. Computar e manter herança múltipla também é trabalho do MI. Esta técnica (às vezes chamada normalização de ontologia) ajuda a manter a ontologia em um estado sustentável e modular.

Isso promove a reutilização da ontologia por outras ontologias e aplicações, mas também minimiza erros humanos inerentes à manutenção de hierarquias com herança múltipla.

Uma vez criada uma definição para **PizzaDeQueijo** (CheesyPizza), pode-se usar o MI para computar automaticamente subclasses de **PizzaDeQueijo** (CheesyPizza).

Exercício 30: Usar o mecanismo de inferência (*Reasoner*) para computar automaticamente a subclasse de **PizzaDeQueijo (CheesyPizza)**

- 1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaDeQueijo** (CheesyPizza).
- 2º. Abrir a aba **Reasoner** (Mecanismo de inferência – MI), para lançar o processo de inferência, clicar em: **Start Reasoner**.

⚠ Verificar os resultados gerados, nos dois painéis de hierarquia **declarada** (*Asserted hierarchy*) e de **hierarquia inferida** (*Inferred hierarchy*), conforme as figuras seguintes:

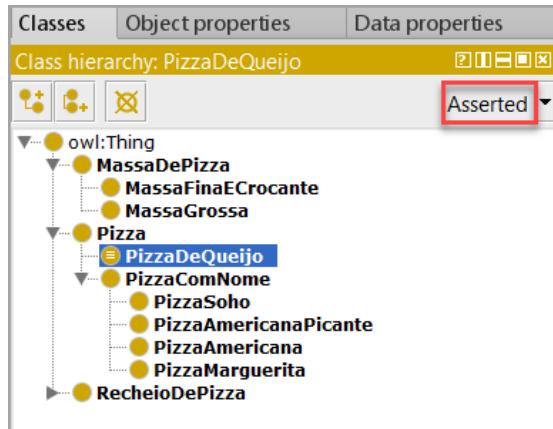


Figura 4.114

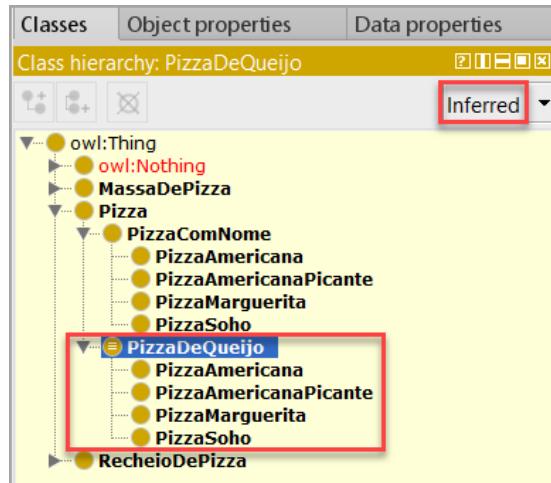


Figura 4.115

O MI determinou que **PizzaMarguerita** (MargheritaPizza), **PizzaAmericana** (AmericanPizza), **PizzaSoho** (SohoPizza) e **PizzaAmericanaPicante** (AmericanHotPizza) são subclasses da **PizzaDeQueijo** (CheesyPizza). Isto ocorreu porque definiu-se **PizzaDeQueijo** (CheesyPizza) usando condições necessárias e suficientes.

Um indivíduo que é uma **Pizza**, e que tem, **pelo menos, um** recheio que é um **RecheioDeQueijo** (CheeseTopping), é membro da classe **PizzaDeQueijo** (CheesyPizza). Como todos os indivíduos descritos pelas classes **PizzaMarguerita** (MargheritaPizza), **PizzaAmericana** (AmericanPizza), **PizzaAmericanaPicante** (AmericanHotPizza) e **PizzaSoho** (SohoPizza) são pizzas e têm, **pelo menos, um** recheio que é um **RecheioDeQueijo** (CheeseTopping) ou recheios que pertencem a subclasses de **RecheioDeQueijo** (CheeseTopping), o MI inferiu que essas classes devem ser subclasses de **RecheioDeQueijo** (CheeseTopping).

! Em geral, as classes não são classificadas como subclasses de Classes Primitivas Primitive Classes (Primitive Classes) pelo MI. As classes primitivas são aquelas que possuem apenas condições necessárias. A exceção a esse fato ocorre quando uma propriedade tem um domínio que é uma classe primitiva. Isto pode fazer com que as classes sejam reclassificadas sob a classe primitiva que é o domínio da propriedade. **Não se recomenda o uso do domínio de propriedade para causar tais efeitos.**

→ Resultados da classificação²⁴

Nas versões anteriores de Protégé (versão 3, por exemplo) (HORRIDGE et al., 2004, p. 64), os relacionamentos superclasse-subclasse inferidos pelo MI podiam ser transferidos para a Hierarquia Declarada (Asserted Hierarchy), a qual foi manualmente construída.

! Apesar da existência desta facilidade, ela **não é considerada uma boa prática para inserir relacionamentos inferidos em hierarquias construídas manualmente** ou na hierarquia declarada enquanto a ontologia está sendo desenvolvida. Por isso

²⁴ Esse item consta na versão 1.0 (HORRIDGE et al., 2004, p. 64), mas foi excluído da versão 1.3 (HORRIDGE; BRANDT, 2011)

aconselha-se evitar o uso dessa funcionalidade durante o desenvolvimento de uma ontologia.

4.12. Restrições universais

Todas as restrições criadas até agora são *restrições existenciais* (\exists ou “some”). Esse tipo de restrição especifica a existência *de, pelo menos, uma* (*some*) relação através de determinada propriedade com um indivíduo membro de uma classe, identificada pelo complemento da restrição (*filler*).

Entretanto, a restrição existencial não obriga que *as únicas* (*only*) relações através da propriedade que possa existir sejam obrigatoriamente com indivíduos membros de uma classe específica (*filler*).

Por exemplo, pode-se usar uma restrição existencial `temRecheio some RecheioDeMuçarela` (ou $\exists \text{ temRecheio RecheioDeMuçarela}$)²⁵ para descrever os indivíduos que têm, pelo menos, uma relação através da propriedade `temRecheio` (`hasTopping`), com um indivíduo membro da classe `RecheioDeMuçarela` (`MozzarellaTopping`). Esta restrição não implica em que todos os tipos de relação `temRecheio` (`hasTopping`) devam ser, obrigatoriamente, membros da classe `RecheioDeMuçarela` (`MozzarellaTopping`).

Para restringir uma relação através de uma propriedade com indivíduos membros de uma classe específica, deve-se usar a restrição universal (Universal Restriction).

As restrições universais (*Universal Restrictions*) são identificadas pelo símbolo \forall ou “only”. Tais restrições condicionam a relação através da propriedade com indivíduos que são membros de uma classe específica.

Por exemplo, a restrição universal expressa pela declaração `temRecheio only RecheioDeMuçarela` (ou $\forall \text{ temRecheio RecheioDeMuçarela}$)²⁶ descreve os indivíduos cuja totalidade das relações do tipo `temRecheio` (`hasTopping`) ocorre com membros da classe `RecheioDeMuçarela` (`MozzarellaTopping`). Os indivíduos não têm relações do tipo `temRecheio` (`hasTopping`) com indivíduos que não são membros da classe `RecheioDeMuçarela` (`MozzarellaTopping`).

Universal restrictions (Restrições universais) são também conhecidas como All Restrictions (Todas as restrições).

A Restrição Universal citada acima `hasTopping temRecheio only RecheioDeMuçarela` (ou $\forall \text{ temRecheio RecheioDeMuçarela}$) também descreve os indivíduos que não participam de nenhuma relação do tipo `temRecheio` (`hasTopping`). Um indivíduo que não participa de nenhuma relação do tipo `temRecheio` (`hasTopping`), por definição, nunca terá relação do tipo `temRecheio` (`hasTopping`) com indivíduos que não são

²⁵ Na versão original em inglês: `hasTopping some MozzarellaTopping` (ou $\exists \text{ hasTopping MozzarellaTopping}$)

²⁶ Na versão original em inglês: `hasTopping only MozzarellaTopping` (ou $\forall \text{ hasTopping MozzarellaTopping}$)

membros da classe **RecheioDeMuçarela** (`MozzarellaTopping`), e a restrição é assim satisfeita.

Para uma determinada propriedade, as *restrições universais (Universals Restictions)* não especificam a existência de relação. Apenas indicam que, se existe uma relação para a propriedade, ele ocorre para indivíduos membros de uma classe.

Por exemplo, deseja-se criar a classe **PizzaVegetariana** (`VegetarianPizza`), de forma que os indivíduos membros dessa classe possam apenas ter recheios que são ou **RecheioDeQueijo** (`CheeseTopping`) ou **RecheioDeVerduras** (`VegetableTopping`). Nesse caso, pode-se usar a restrição universal.

Exercício 31: Criar uma classe para descrever uma PizzaVegetariana (VegetarianPizza)

1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **Pizza**.

2º. Criar uma subclasse  **PizzaVegetariana** (`VegetarianPizza`) à classe **Pizza**.

3º. Selecionar a nova classe **PizzaVegetariana** (`VegetarianPizza`) e clicar no ícone  **SubClassOf** (`SubClasseDe`), no painel **Classes Description** (Descrição de classes).

4º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição: `temRecheio only (RecheioDeQueijo or RecheioDeVerduras)`

 (1) propriedade da restrição sobre a classe Pizza: “**temRecheio**”; (2) o tipo de restrição: “**only**” (restrição universal); (3) o complemento (obrigatório) da restrição (*Filler*): a união das classes “**RecheioDeQueijo**” e “**RecheioDeVerduras**”, com o operador “**or**”.

5º. Após digitar a restrição completa (com a sintaxe correta), conforme a figura seguinte, clicar em **OK**:

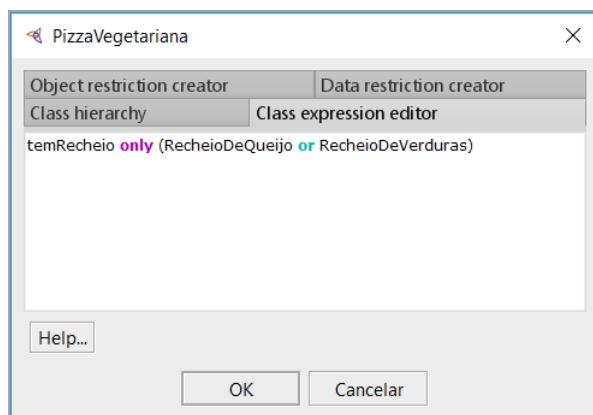


Figura 4.116

 Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:

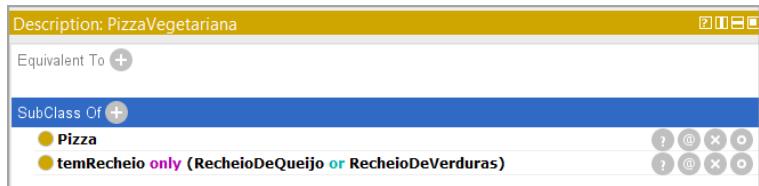


Figura 4.117

Se alguma coisa é membro da classe **PizzaVegetariana** (VegetarianPizza) é necessário que ela seja uma **Pizza**, e é necessário que tenha apenas recheios que são tipos de **RecheioDeQueijo** (CheeseTopping) ou tipos de **RecheioDeVerduras** (VegetableTopping). Em outras palavras, todos as relações do tipo **temRecheio** (hasTopping) dos quais os indivíduos membros de **PizzaVegetariana** (VegetarianPizza) participam, devem ser com indivíduos que são membros das classes **RecheioDeQueijo** (CheeseTopping) ou **RecheioDeVerduras** (VegetableTopping). A classe **PizzaVegetariana** (VegetarianPizza) também contém indivíduos que são Pizzas e não participam em nenhuma relação do tipo **temRecheio** (hasTopping).

Em situações como a do exemplo acima, um erro comum é usar a interseção ao invés da união. Por exemplo, **RecheioDeQueijo** \cap **RecheioDeVerduras**²⁷ é lido como **RecheioDeQueijo AND RecheioDeVerduras**²⁸. Embora **RecheioDeQueijo AND RecheioDeVerduras** seja uma frase comum em linguagem natural, em termos lógicos significa que algo é simultaneamente um tipo de **RecheioDeQueijo** (CheeseTopping) e de **PizzaVegetariana** (VegetarianPizza). Isto é incorreto conforme demonstrado na seção 4.9.2. (p. 63). Se a classe **RecheioDeQueijo** (CheeseTopping) e **PizzaVegetariana** (VegetarianPizza) não são disjuntas, é possível dizer isso com legitimidade lógica. Dessa forma, a classe não é inconsistente e, portanto, não será destacada por um MI.

No exemplo acima, uma opção é criar duas restrições universais, uma para **RecheioDeQueijo** (CheeseTopping) (**temRecheio only RecheioDeQueijo** ou \forall **temRecheio RecheioDeQueijo**)²⁹ e outra para **RecheioDeVerduras** (VegetableTopping) (**temRecheio only RecheioDeVerduras** ou \forall **temRecheio RecheioDeVerduras**)³⁰.

Entretanto, quando as restrições múltiplas são utilizadas (para qualquer tipo de restrição), a **descrição** total é considerada como a interseção das restrições individuais. Isso é equivalente a uma restrição com um complemento da restrição (*filler*) que é a interseção de **RecheioDeMuçarela** (MozzarellaTopping) e **RecheioDeTomate** (TomatoTopping). Conforme explicado, isto é incorreto do ponto de vista lógico.

No momento, **PizzaVegetariana** (VegetarianPizza) é descrita usando condições necessárias. Entretanto, a descrição de **PizzaVegetariana** (VegetarianPizza) poderia ser considerada

²⁷ Na versão original em inglês: CheeseTopping \cap VegetableTopping

²⁸ Na versão original em inglês: CheeseTopping AND VegetableTopping

²⁹ Na versão original em inglês: (hasTopping only CheeseTopping ou \forall hasTopping CheeseTopping)

³⁰ Na versão original em inglês: (hasTopping only VegetableTopping ou \forall hasTopping VegetableTopping)

como completa. Sabe-se que qualquer indivíduo que satisfaça a essas condições deve ser um **PizzaVegetariana** (VegetarianPizza).

Pode-se, portanto, converter as condições necessárias de **PizzaVegetariana** (VegetarianPizza) em condições necessárias e suficientes. Isto vai permitir usar o MI para determinar a subclasse de **PizzaVegetariana** (VegetarianPizza).

Exercício 32: Converter as condições necessárias da classe **PizzaVegetariana** (VegetarianPizza) em condições necessárias e suficientes

- 1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaVegetariana** (VegetarianPizza).
- 2º. Abrir a aba **Edit** e selecionar a função **Convert to defined class** (Converter em uma classe definida).

⚠ Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte (comparar o resultado com a Figura 4.117, acima):

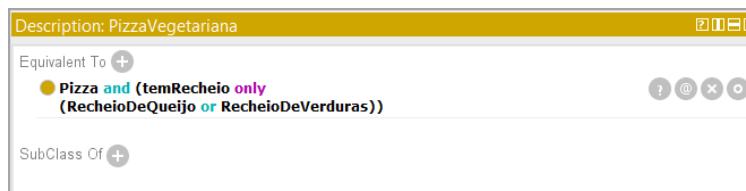


Figura 4.118

Converteu-se a **definição** de **PizzaVegetariana** (VegetarianPizza) em uma **definição**.

Se alguma coisa é uma **PizzaVegetariana** (VegetarianPizza), então é necessário que seja uma Pizza, também é necessário que todos os recheios pertençam à classe **RecheioDeQueijo** (CheeseTopping) ou **RecheioDeVerduras** (VegetableTopping). Além disso, se alguma coisa é membro da classe Pizza e todos os recheios são membros da classe **RecheioDeQueijo** (CheeseTopping) ou **RecheioDeVerduras** (VegetableTopping), essas condições são suficientes para reconhecer que tal coisa é um membro da classe **PizzaVegetariana** (VegetarianPizza).

4.13. Classificação automática e Raciocínio de Mundo Aberto (Open World Reasoning – OWR)

Deseja-se usar o MI para computar automaticamente o relacionamento superclasse-subclasse (*subsumption relationship*) entre **PizzaMarguerita** (MargheritaPizza) e **PizzaVegetariana** (VegetarianPizza), e entre **PizzaSoho** (SohoPizza) e **PizzaVegetariana** (VegetarianPizza). Acredita-se que **PizzaMarguerita** (MargheritaPizza) e **PizzaSoho** (SohoPizza) devam ser pizzas vegetarianas (subclasses de **PizzaVegetariana**) porque têm recheios vegetarianos: pela definição, recheios vegetarianos são membros das classes **RecheioDeQueijo** (CheeseTopping) ou **RecheioDeVerduras** (VegetableTopping) e de suas

subclasses. Tendo anteriormente criado uma definição para **PizzaVegetariana** (VegetarianPizza) usando um conjunto de condições necessárias e suficientes, pode-se usar o MI para classificação automática e para determinar as pizzas que são **PizzaVegetariana** (VegetarianPizza) na ontologia.

Exercício 33: Usar o mecanismo de inferência (*Reasoner*) para computar automaticamente a classe PizzaVegetariana (VegetarianPizza)

1º. Abrir a aba **Reasoner** (Mecanismo de inferência – MI), para lançar o processo de inferência, clicar em: **Start Reasoner**.

⚠ Verificar os resultados gerados, no painel de **hierarquia inferida** (*Inferred hierarchy*), conforme a figura seguinte:

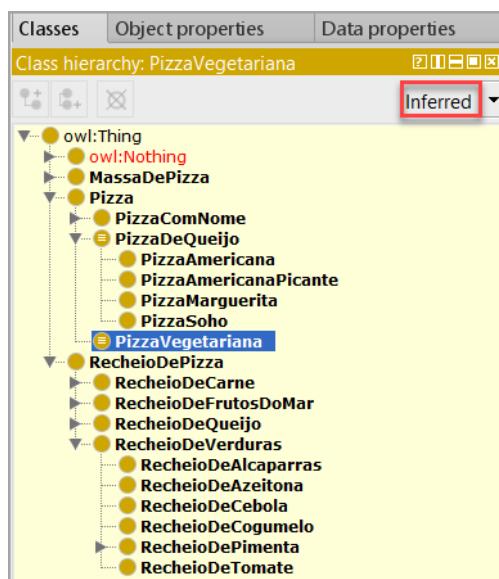


Figura 4.119

Observe-se, na Figura 4.119, que **PizzaMarguerita** (MargheritaPizza) e **PizzaSoho** (SohoPizza) não foram classificadas como subclasses de **PizzaVegetariana** (VegetarianPizza). Pode parecer estranho, pois **PizzaMarguerita** (MargheritaPizza) e **PizzaSoho** (SohoPizza) têm ingredientes vegetarianos, ou seja, ingredientes que são tipos de **RecheioDeQueijo** (CheeseTopping) ou de **RecheioDeVerduras** (VegetableTopping).

No entanto, pode-se observar que **PizzaMarguerita** (MargheritaPizza) e **PizzaSoho** (SohoPizza) **perderam algo da definição**, e assim não podem ser classificadas como subclasses de **PizzaVegetariana** (VegetarianPizza).

As inferências em OWL (Lógica Descritiva) se baseiam na Open World Assumption (OWA) (Postulado de Mundo Aberto), também conhecida como Open World Reasoning (OWR) (Raciocínio de Mundo Aberto). O Postulado de Mundo Aberto significa que não se pode assumir que alguma coisa não existe, até que seja estabelecido explicitamente que ela não existe. Em outras palavras, porque alguma coisa não foi definida como verdade, não

significa que ela é falsa: presume-se que tal conhecimento apenas não foi adicionado à base de conhecimento.

No caso da ontologia de Pizza, indica-se que **PizzaMarguerita** (MargheritaPizza) tem recheios que são tipos de **RecheioDeMuçarela** (MozzarellaTopping) e também tipos de **RecheioDeTomate** (TomatoTopping). Por causa do Postulado de Mundo Aberto, até que se diga explicitamente que uma **PizzaMarguerita** (MargheritaPizza) tem apenas esses dois tipos de recheios, presume-se (pelo MI) que uma **PizzaMarguerita** (MargheritaPizza) pode ter outros recheios.

Para especificar explicitamente que uma **PizzaMarguerita** (MargheritaPizza) tem recheios que são tipos de **RecheioDeMuçarela** (MozzarellaTopping) ou tipos de **RecheioDeTomate** (TomatoTopping), e apenas (*only*) tipos de **RecheioDeMuçarela** (MozzarellaTopping) ou tipos de **RecheioDeTomate** (TomatoTopping), deve-se adicionar um **Closure Axiom (Axioma de Fechamento)** à propriedade **temRecheio** (hasTopping). Um axioma de fechamento (Closure Axiom) também é chamado de Closure Restriction (Restrição de Fechamento).

4.13.1. Axiomas de fechamento (*Closure axiom*)

Um axioma de fechamento (Closure Axiom) consiste em uma restrição universal que atua na propriedade informando que ela pode apenas ser preenchida por complementos de restrição (*fillers*) específicos. A restrição tem um complemento de restrição (*filler*) que é a união dos complementos de restrição (*fillers*) que ocorrem nas restrições existenciais da propriedade.

Por exemplo, o axioma de fechamento (Closure Axiom) da propriedade **temRecheio** (hasTopping) para **PizzaMarguerita** (MargheritaPizza) é uma restrição universal que atua na propriedade **temRecheio** (hasTopping), com um complemento de restrição (*filler*) que é a união de **RecheioDeMuçarela** (MozzarellaTopping) e de **RecheioDeTomate** (TomatoTopping), ou seja, a declaração: $\text{temRecheio} \text{ only } (\text{RecheioDeMuçarela} \cup \text{RecheioDeTomate})$ (ou $\forall \text{ temRecheio } (\text{RecheioDeMuçarela} \cup \text{RecheioDeTomate})$)³¹.

Exercício 34: Adicionar um axioma de fechamento (Closure axiom) à propriedade **temRecheio** (hasTopping) para **PizzaMarguerita** (MargheritaPizza)

- 1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaMarguerita** (MargheritaPizza).
- 2º. Clicar no ícone  **SubClassOf** (SubClasseDe), no painel **Classes Description** (Descrição de classes).

³¹ Na versão original em inglês: $\text{hasTopping} \text{ only } (\text{MozzarellaTopping} \cup \text{TomatoTopping})$ (ou $\forall \text{ hasTopping } (\text{MozzarellaTopping} \cup \text{TomatoTopping})$)

3º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição: `temRecheio only (RecheioDeMuçarela or RecheioDeTomate)`³²

⚠ (1) propriedade da restrição sobre a classe Pizza: “**temRecheio**”; (2) o tipo de restrição: “**only**” (restrição universal); (3) o complemento (obrigatório) da restrição (*Filler*): a união das classes “**RecheioDeMuçarela**” e “**RecheioDeTomate**”, com o operador “**or**”.

4º. Após digitar a restrição completa (com a sintaxe correta), conforme a figura seguinte, clicar em **OK**:

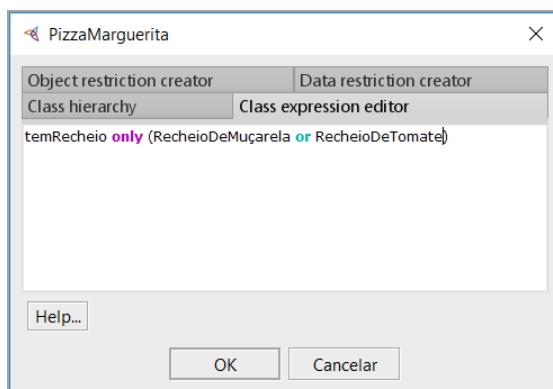


Figura 4.120

⚠ Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:

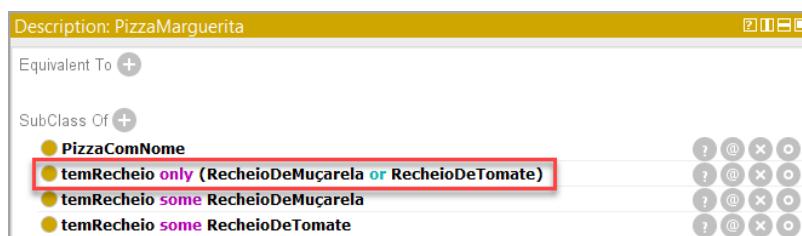


Figura 4.121

Isto significa que um indivíduo membro da classe **PizzaMarguerita** (`MargheritaPizza`) deve ser membro da classe **Pizza**, deve ter, **pelo menos, um** recheio que é um tipo de **RecheioDeMuçarela** (`MozzarellaTopping`), deve ter, pelo menos, um recheio que é membro de **RecheioDeTomate** (`TomatoTopping`), e os recheios devem ser apenas tipos de **RecheioDeMuçarela** (`MozzarellaTopping`) ou **RecheioDeTomate** (`TomatoTopping`).

⚠ Um erro comum é usar apenas restrições universais nas descrições. Por exemplo, descreve-se **PizzaMarguerita** (`MargheritaPizza`) como subclasse de **Pizza**, usando apenas a declaração `temRecheio only (RecheioDeMuçarela ∪ RecheioDeTomate)` ou \forall `temRecheio (RecheioDeMuçarela ∪ RecheioDeTomate)`³³ sem nenhuma restrição existencial (“**some**”, foram mantidas na

³² Com símbolos lógicos: \forall `temRecheio (RecheioDeMuçarela ∪ RecheioDeTomate)`

³³ Em inglês: \forall `hasTopping (MozzarellaTopping ∪ TomatoTopping)`

Figura 4.121, acima). Entretanto, pela semântica da restrição universal, a declaração significa realmente: “coisas que são Pizzas e somente têm recheios que são RecheioDeMuçarela (MozzarellaTopping) ou RecheioDeTomate (TomatoTopping), OU, coisas que são Pizzas e não têm qualquer recheio”.

Exercício 35: Adicionar um axioma de fechamento (Closure axiom) à propriedade temRecheio (hasTopping) para PizzaSoho (SohoPizza)

- 1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaSoho** (SohoPizza).
- 2º. Clicar no ícone **SubClassOf** (SubClasseDe), no painel **Classes Description** (Descrição de classes).
- 3º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição: **temRecheio only** (RecheioDeMuçarela **or** RecheioDeTomate **or** RecheioDeParmesão **or** RecheioDeAzeitona)
- 4º. Após digitar a restrição completa (com a sintaxe correta), conforme a figura seguinte, clicar em **OK**:

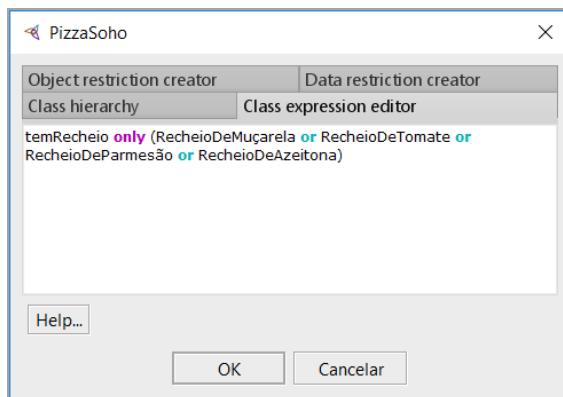


Figura 4.122

Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:

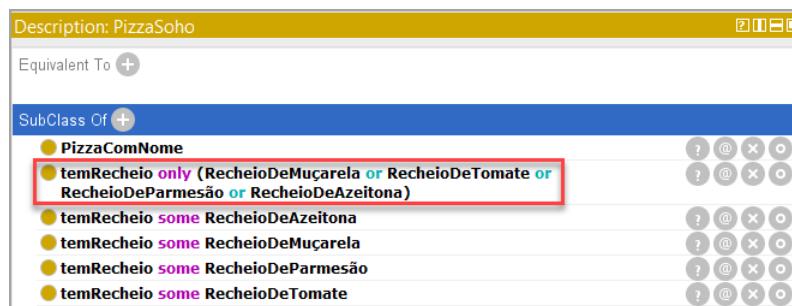


Figura 4.123

Para complementar, adicionar um axioma de fechamento (Closure Axiom) à propriedade **temRecheio** (hasTopping) para as classes **PizzaAmericana** (AmericanPizza) e **PizzaAmericanaPicante** (AmericanHotPizza). A atividade de inserir manualmente axiomas

de fechamento parece trabalhosa, mas o Protégé 5 possui recursos para criá-los automaticamente.

Exercício 36: Criar automaticamente um axioma de fechamento na propriedade temRecheio (hasTopping) para a classe PizzaAmericana (AmericanPizza)

1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaAmericana** (AmericanPizza).

2º. No painel **Classes Description** (Descrição de classes), clicar no botão direito do mouse no ícone de uma das restrições de **SubClassOf** (SubClasseDe), por exemplo, **temRecheio some RecheioDeTomate**

Aparece, em seguida, um menu suspenso com várias opções. Clicar em: “**Create closure axiom**” (Criar um axioma de fechamento):

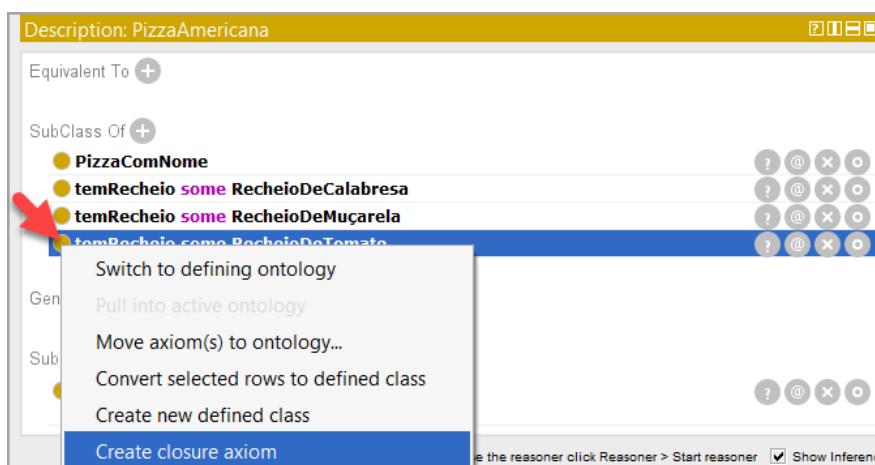


Figura 4.124

Verificar o resultado gerado automaticamente, no painel **Description**, conforme a figura seguinte:

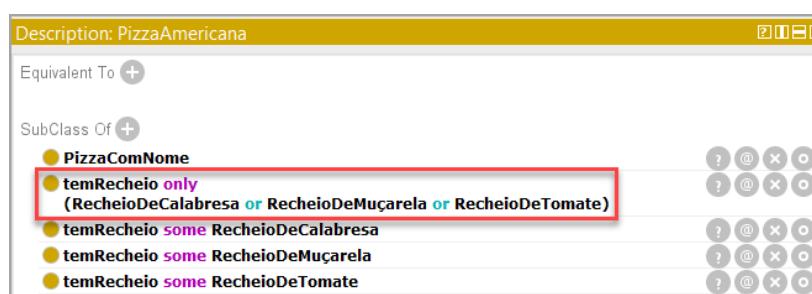


Figura 4.125

Exercício 37: Criar automaticamente um axioma de fechamento na propriedade temRecheio (hasTopping) para a classe PizzaAmericanaPicante (AmericanHotPizza)

1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaAmericanaPicante** (AmericanHotPizza).

2º. No painel **Classes Description** (Descrição de classes), clicar no botão direito do mouse no ícone de uma das restrições de **SubClassOf** (SubClasseDe), por exemplo, **temRecheio some RecheioDeTomate**

Aparece, em seguida, um menu suspenso com várias opções. Clicar em: “**Create closure axiom**” (Criar um axioma de fechamento):

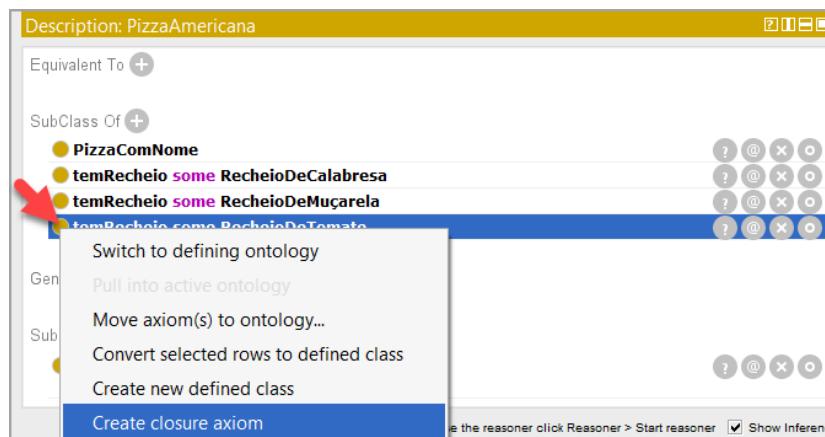


Figura 4.126

Verificar o resultado gerado automaticamente, no painel **Description**, conforme a figura seguinte:

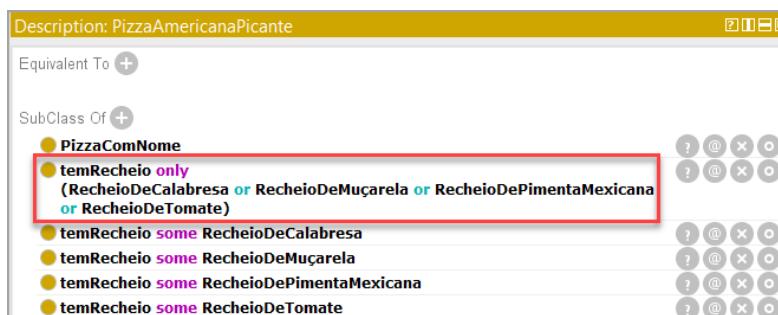


Figura 4.127

Tendo adicionado axiomas de fechamento na propriedade **temRecheio** (hasTopping) para as pizzas, usa-se agora o MI para computar automaticamente a classificação.

Exercício 38: Usar o mecanismo de inferência (**Reasoner**) para computar as subclasses de **PizzaDeQueijo** e de **PizzaVegetariana**

1º. Abrir a aba **Reasoner** (Mecanismo de inferência – MI), para lançar o processo de inferência, clicar em: **Start Reasoner**.

Verificar os resultados gerados, no painel de **hierarquia inferida** (*Inferred hierarchy*), conforme a figura seguinte:

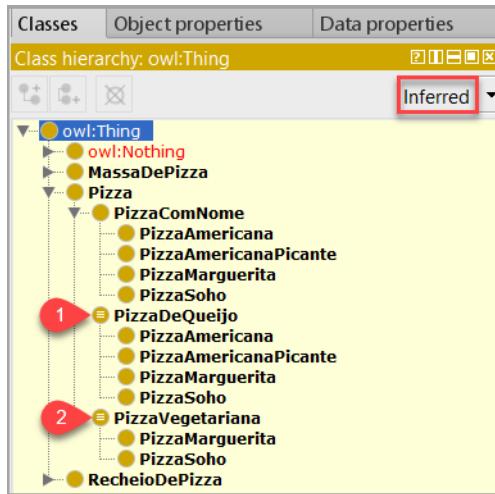


Figura 4.128

A ontologia é classificada e os resultados são apresentados no painel *Inferred Hierarchy* (Hierarquia Inferida) (Figura 4.128, acima). Desta vez, **PizzaMarguerita** (MargheritaPizza) e **PizzaSoho** (SohoPizza) estão classificadas como subclasses de **PizzaVegetariana** (VegetarianPizza)❷. Isto ocorreu porque especificou-se a propriedade **temRecheio** (hasTopping) como Closed (Fechada) nas pizzas, para dizer exatamente quais recheios elas têm. Além disso, **PizzaVegetariana** (VegetarianPizza) foi definida para ser uma Pizza apenas com tipos de **RecheioDeQueijo** (CheeseTopping) e de **RecheioDeVerduras** (VegetableTopping).

Embora a ontologia em questão seja uma estrutura simples neste estágio, o uso do MI pode auxiliar, especialmente no caso de grandes ontologias, a manter a hierarquia de herança múltipla.

4.14. Partições de valores (Value Partitions)

Nessa seção criam-se *value partitions* (partições de valor), as quais são usadas para refinar as descrições de classes. As partições de valor (*value partitions*) não são partes da linguagem OWL, ou de outra linguagem de ontologia, são um *padrão de projeto* (*design pattern*).

Os padrões de projetos em ontologias são similares a padrões de projeto em programação orientada a objetos (POO): são soluções desenvolvidas por especialistas e reconhecidos como soluções para problemas comuns de modelagem.

Conforme mencionado, as partições de valor podem ser criadas para refinar descrições de classe. Por exemplo, cria-se uma partição de valor chamada **SpicinessValuePartition** (TeorPicante) para descrever o picante (*spiciness*) de **PizzaToppings** (RecheiosDePizza).

As partições de valor restringem a faixa de valores possíveis para uma lista exaustiva, por exemplo, a SpicinessValuePartition restringe a faixa para **Mild** (Levemente apimentado), **Medium** (Médio), e **Hot** (Muito Apimentado).

Para criar uma partição de valor em OWL são necessários alguns passos:

- 1º. Criar uma classe para representar a partição de valor. Por exemplo, para representar a partição de valor do picante (spiciness), cria-se a classe **TeorPicante** (SpicinessValuePartition).
- 2º. Criar subclasses da partição para representar as opções. Por exemplo, pode-se criar as classes **Leve** (Mild) (Levemente apimentado), **Moderado** (Medium) e **Forte** (Hot) (Muito Apimentado) como subclasses da classe **TeorPicante** (SpicinessValuePartition).
- 3º. Tornar essas subclasses disjuntas.
- 4º. Fornecer um axioma de cobertura (Covering Axiom), de forma que a lista de valores seja exaustiva.
- 5º. Criar uma propriedade objeto (Object Property) para a partição de valor. Por exemplo, para a SpicinessValuePartition pode-se criar a propriedade temPicante (hasSpiciness).
- 6º. Tornar essa propriedade *funcional*.
- 7º. Defina o escopo (range) da propriedade como a classe **PartiçãoDeValor** (ValuePartition). Por exemplo, para a propriedade **temPicante** (hasSpiciness) o escopo é definido como **TeorPicante** (SpicinessValuePartition).

Deseja-se criar uma **PartiçãoDeValor** (ValuePartition) para descrever o tempero dos recheios de Pizza. Então será possível classificar as pizzas em pizzas apimentadas e em pizzas não apimentadas. Será possível também dizer que os recheios das pizzas têm um tanto de pimenta que as classifique como: **Leve** (Mild) (Levemente apimentado), **Moderado** (Medium) (Médio) e **Forte** (Hot) (Muito Apimentado).

Note-se que tais opções são mutuamente exclusivas: alguma coisa não pode ser, ao mesmo tempo, **Moderado** (Medium) (Médio) e **Forte** (Hot) (Muito Apimentado), ou uma combinação de opções.

Exercício 39: Criar uma partição de valor (Value Partition) para representar o picante (spiciness) em recheios de Pizza

- 1º. Na aba **Class hierarchy** (Hierarquia de classe), criar uma classe **PartiçãoDeValor** (ValuePartition), sendo uma subclasse da classe principal **owl:Thing**.
 - 2º. Criar uma classe **TeorPicante** (SpicinessValuePartition), subclasse de **PartiçãoDeValor** (ValuePartition).
 - 3º. Criar 3 classes **Forte** (Hot), **Moderado** (Medium), **Leve** (Mild), subclasses de **TeorPicante** (SpicinessValuePartition).
-  Criar essas classes com o editor de hierarquia: Selecionar **owl:Thing** e abrir a aba **Tools** (Ferramentas), no menu superior de Protégé 5, e selecionar **Create Class hierarchy...** (Criar Hierarquia de Classe...) (ver Quadro 4.1, p.27):

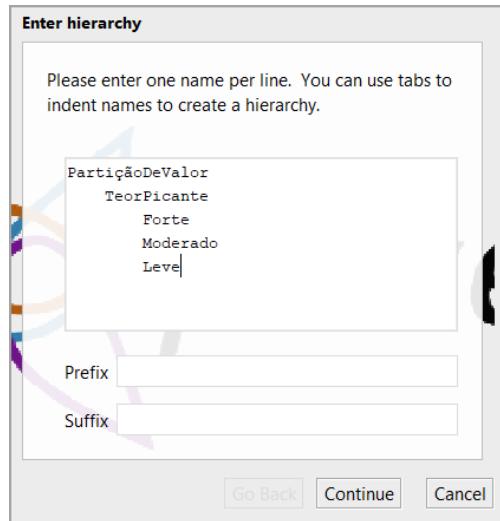


Figura 4.129

⚠ Verificar o resultado gerado automaticamente, no painel Class hierarchy (Hierarquia de classe), conforme a figura seguinte:

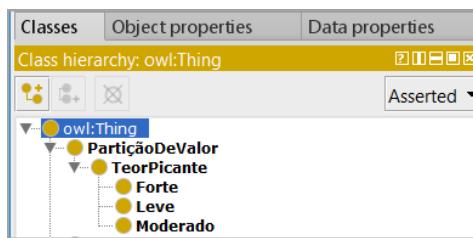


Figura 4.130

4º. Tornar disjuntas classes **Forte** (Hot), **Moderado** (Medium), **Leve** (Mild). Clicando com o botão direito do mouse sobre uma das três classes, por exemplo, **Forte** (Hot). Aparece um menu suspenso, clicar em: **Make primitive siblings disjoint** (Tornar disjuntas as classes primitivas irmãs)

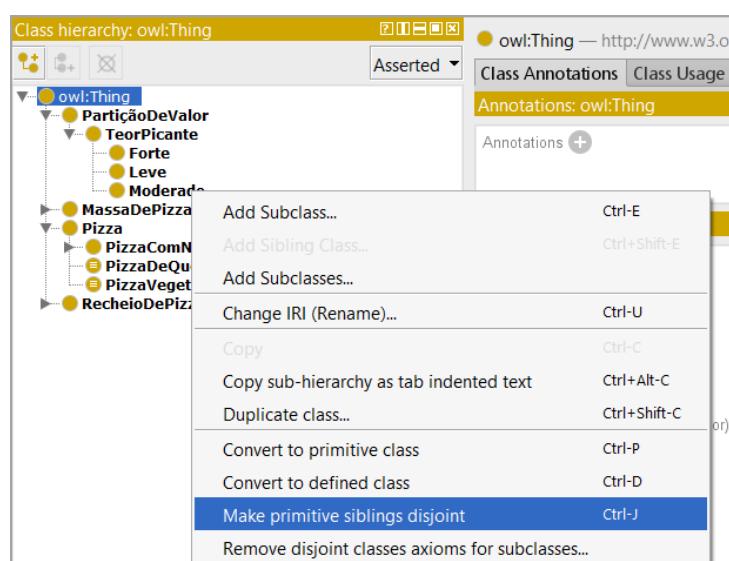


Figura 4.131

⚠ Verificar o resultado gerado, no painel **Description**, por exemplo, com a classe **Forte** (Hot), conforme a figura seguinte:



Figura 4.132

- 5º. Na aba **Object Property** (Propriedade de objeto), criar uma propriedade **temPicante** (**hasSpiciness**)❶, sendo uma subclasse da classe principal **owl:topObjectProperty**.
- 6º. Clicar no ícone **+ Ranges** (Escopos), no painel **Properties Description** (Descrição de propriedades), para definir o escopo da propriedade: **TeorPicante** (**SpicinessValuePartition**)❷.
- 7º. Selecionar a marca **Functional** ❸, para tornar funcional a propriedade **temPicante** (**hasSpiciness**), conforme a figura seguinte:

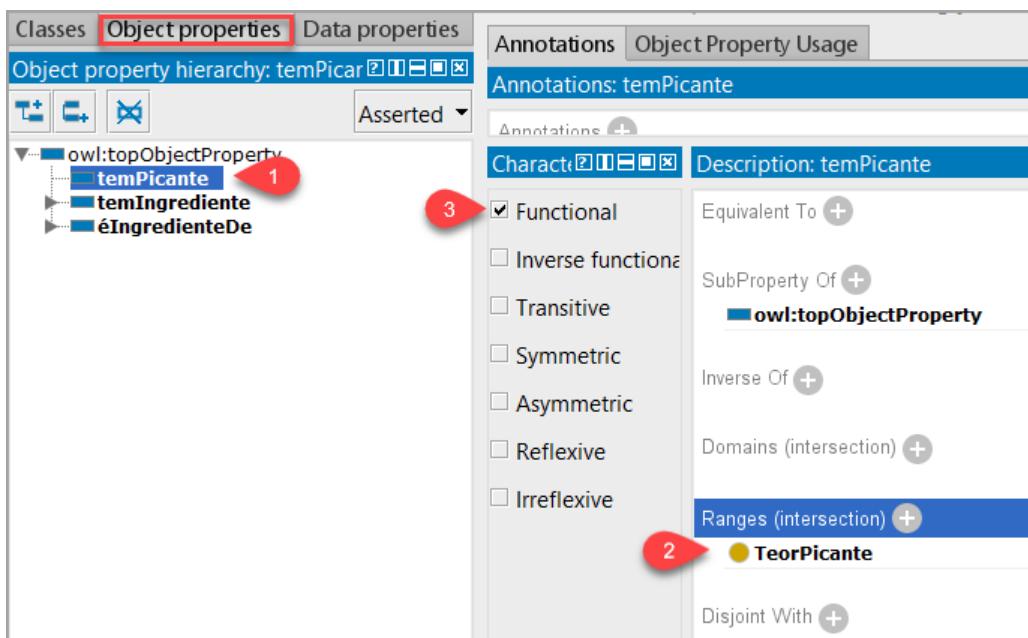


Figura 4.133

- 8º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **TeorPicante** (**SpicinessValuePartition**).
- 9º. Clicar no ícone **+ EquivalentTo** (Equivalente a), no painel **Class Description** (Descrição de classe), para criar um **axioma de cobertura** (Covering axiom): **Forte** (Hot) ou **Moderado** (Medium) ou **Leve** (Mild).

10º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição: Forte **or** Moderado **or** Leve

11º. Após digitar a restrição completa (com a sintaxe correta), conforme a figura seguinte, clicar em **OK**:

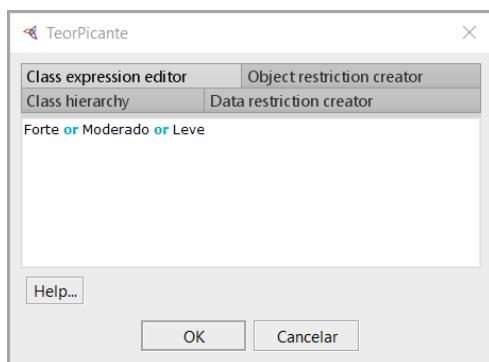


Figura 4.134

⚠ Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:



Figura 4.135

4.14.1. Axiomas de cobertura (Covering axioms)

Como parte do padrão partição de valor (ValuePartition), usou-se um Covering Axiom (Axioma de Cobertura), o qual consiste de duas partes: a classe que está sendo coberta, e as classes que formam a cobertura.

Por exemplo, tem-se três classes A, B e C, e as classes B e C são subclasses de A. Tem-se um axioma de cobertura (Covering Axiom) que especifica que a classe A é coberta pela classe B e também pela classe C. Isto significa que um membro da classe A deve ser membro da classe B e/ou C. Se as classes B e C são disjuntas, então um membro de A deve ser um membro de B ou de C. Em geral, embora B e C sejam subclasses de A, um indivíduo pode ser um membro de A sem ser membro de uma das classes B ou C.

No Protégé 5, um axioma de cobertura manifesta-se como uma classe que é a união das classes que estão sendo cobertas, as quais formam a superclasse da classe que está sendo coberta. No caso de A, B e C, a classe A pode ter uma subclasse de $B \cup C$ (ou $B \text{ or } C$).

O efeito de um axioma de cobertura é representado na Figura 4.136 seguinte:

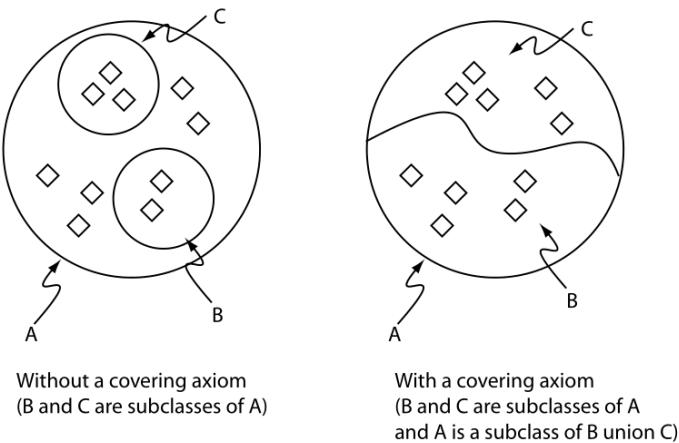


Figura 4.136 – Um exemplo de axioma de cobertura

A Figura 4.136 mostra, de forma esquematizada, o efeito de usar um axioma de cobertura (Covering Axiom) para cobertura da classe A com as classes B e C.

A classe **TeorPicante** (SpicinessValuePartition) tem um axioma de cobertura para indicar sua cobertura pelas classes **Leve** (Mild), **Moderado** (Medium) e **Forte** (Hot), as quais são disjuntas para que um indivíduo não possa ser um membro de mais de uma delas. A classe **TeorPicante** (SpicinessValuePartition) tem uma superclasse que é **Leve or Moderado or Forte**³⁴. A cobertura indica que um membro de **TeorPicante** (SpicinessValuePartition) deve ser membro de uma das classes: **Leve** (Mild) ou **Moderado** (Medium) ou **Forte** (Hot).

A diferença entre usar ou não um axioma de cobertura é representada na Figura 4.137. Em ambos os casos, as classes **Leve** (Mild), **Moderado** (Medium) e **Forte** (Hot) são disjuntas, elas não se sobrepõem.

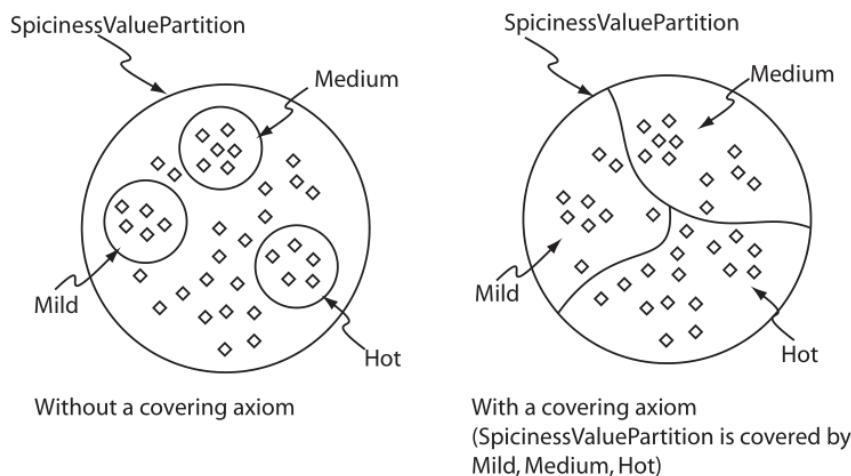


Figura 4.137 – O efeito do axioma de cobertura sobre a classe SpicinessValuePartition (TeorPicante)

No caso de não utilização do axioma de cobertura, um indivíduo pode ser um membro da classe **TeorPicante** (SpicinessValuePartition) e não ser membro das classes **Leve** (Mild),

³⁴ Em inglês: $\text{Mild} \cup \text{Medium} \cup \text{Hot}$ (ou $\text{Mild or Medium or Hot}$)

Moderado (Medium) e **Forte** (Hot), uma vez que **TeorPicante** (SpicinessValuePartition) não é coberta pelas classes **Leve** (Mild), **Moderado** (Medium) e **Forte** (Hot).

Comparando com o caso em que a cobertura de axioma é usada, se um indivíduo é membro da classe **TeorPicante** (SpicinessValuePartition), ele deve ser membro de uma das três subclasses **Leve** (Mild), **Moderado** (Medium) e **Forte** (Hot), uma vez que **TeorPicante** (SpicinessValuePartition) é coberta pelas classes **Leve** (Mild), **Moderado** (Medium) e **Forte** (Hot).

4.15. Acrescentar picante aos recheios de pizzas³⁵

Pode-se usar a classe **TeorPicante** (SpicinessValuePartition) para descrever os temperos dos recheios de pizza. Para fazer isto, é necessário adicionar uma restrição existencial para cada tipo de **RecheiodePizza** (PizzaTopping), de forma a indicar que ela é apimentada.

A forma da restrição é `temPicante some TeorPicante`³⁶, onde **TeorPicante** (SpicinessValuePartition) é **Leve** (Mild), **Moderado** (Medium) ou **Forte** (Hot).

Exercício 40: Especificar a propriedade restritiva temPicante (hasSpiciness) nas classes de recheios de pizzas

- 1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **RecheioDePimentaMexicana** (JalapenoPepperTopping)
- 2º. Clicar no ícone **EquivalentTo** (Equivalente a), no painel **Class Description** (Descrição de classe), para criar uma restrição existencial (*Existential restriction*) (*some*): **temPicante** (**hasSpiciness**) *some* **Forte** (Hot).
- 3º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição: **temPicante some Forte**
- 4º. Após digitar a restrição completa (com a sintaxe correta), conforme a figura seguinte, clicar em **OK**:

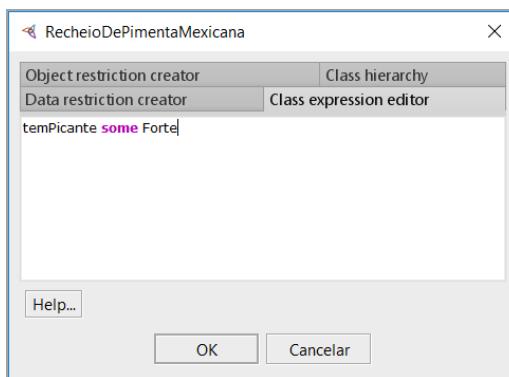


Figura 4.138

³⁵ Na versão 1.0 (HORRIDGE et al., 2004, p. 77–79), foi usado o aplicativo *Property Matrix Wizard* (Assistente Matriz de Propriedade) para realizar essa tarefa. Na versão 1.3 (HORRIDGE; BRANDT, 2011, p. 70–72), um aplicativo “Matrix Plugin” é apresentado no Apêndice C (p. 104–107).

³⁶ Em inglês: `hasSpiciness some SpicinessValuePartition` (ou $\exists \text{ hasSpiciness SpicinessValuePartition}$)



Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:

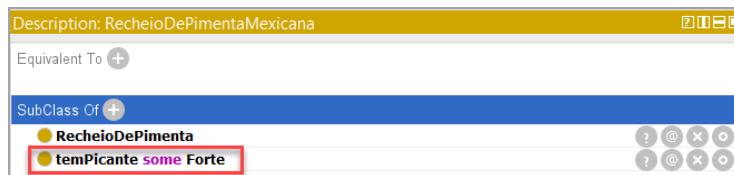


Figura 4.139

- 5º. Repetir esse processo para cada **RecheioDePizza** (PizzaTopping). Para agilizar o processo³⁷: seguir as etapas acima, na 3ª, a restrição pode ser colada no editor. Caso precisar, só mudar o valor do TeorPicante (o *filler* da expressão). Usar a tecla **[Tab]** para autocompletar as expressões.



Algumas orientações para escolher o TeorPicante:

Hierarquia de **RecheioDePizza** (PizzaTopping) e seu **TeorPicante** (SpicinessValuePartition):

Queijo

Muçarela => Leve

Parmesão => Leve

Carne

Presunto => Leve

Calabresa => Moderado

Salame => Moderado

CarneApimentada => Forte

FrutosDoMar

Anchova => Leve

Camarão => Leve

Atum => Leve

Verduras

Alcaparras => Leve

Cogumelo => Leve

Azeitona => Leve

Cebola => Leve

Pimenta

PimentaVermelha => Forte

PimentaVerde => Forte

PimentaMexicana => Forte

Tomate => Leve

Para completar esta seção, criando uma classe **PizzaPicante** (SpicyPizza), que deve ter Pizzas com recheios apimentados como subclasses. Para fazer isto define-se a classe **PizzaPicante** (SpicyPizza) como uma Pizza com pelo menos um **RecheioDePizza** (PizzaTopping) e um tempero apimentado (**temPicante** | **hasSpiciness**) que é **Forte** (Hot). Isto pode ser feito de mais de uma maneira: criando uma restrição na propriedade **RecheioDePizza** (PizzaTopping) que tem uma restrição sobre a propriedade **temPicante** (**hasSpiciness**) como seu complemento (obrigatório) da restrição (*Filler*).

³⁷ Na versão 1.3 (HORRIDGE; BRANDT, 2011, p. 104–107), um aplicativo “Matrix Plugin” é apresentado no Apêndice C para realizar esse processo de forma mais ágil e com menos riscos de erros.

Exercício 41: Criar uma classe PizzaPicante (SpicyPizza) como uma subclasse de Pizza

- 1º. Criar uma classe **PizzaPicante** (SpicyPizza), subclasse de **Pizza**.
 - 2º. Selecionar a classe **PizzaPicante** (SpicyPizza).
 - 3º. Clicar no ícone  **SubClassOf** (SubClasseDe), no painel **Classes Description** (Descrição de classes).
 - 4º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição: `temRecheio some (RecheioDePizza and (temPicante some Forte))`
- ⚠️** O complemento (obrigatório) da restrição (*Filler*) deve ser: `RecheioDePizza (PizzaTopping) and temPicante (hasSpiciness) some Forte (Hot)`. Este complemento da restrição (*filler*) descreve uma classe anônima, a qual contém os indivíduos que são membros da classe `RecheioDePizza (PizzaTopping)` e também membros da classe de indivíduos relacionados aos membros da classe `Forte (Hot)` através da propriedade `temPicante (hasSpiciness)`. Em outras palavras, diz respeito às coisas que são `RecheioDePizza (PizzaTopping)` e tem um tempero apimentado (`spiciness`) que é `Forte (Hot)`. Para entrar com esta restrição, só digitar no campo de edição de expressões a declaração (em inglês): `(PizzaTopping and (hasSpiciness some Hot))` ou em português: `(RecheioDePizza and (temPicante some Forte))`
- 5º. Após digitar a restrição completa, clicar em **OK**:

⚠️ Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:



Figura 4.140

- 3º. Para finalizar, selecionar a classe **PizzaPicante** (SpicyPizza).
 - 4º. Abrir a aba **Edit** e selecionar a função **Convert to defined class** (Converter em uma classe definida).
- 💡** Essa função é também acessível clicando no botão direito do mouse sobre uma classe, por exemplo, **PizzaPicante** (SpicyPizza):

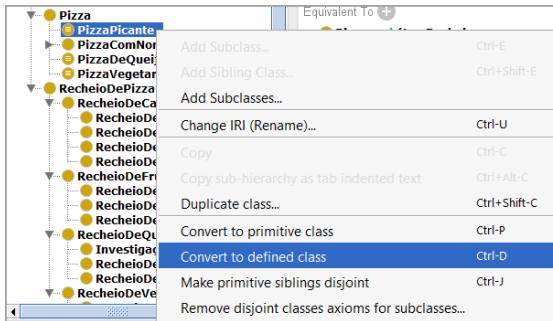


Figura 4.141

⚠ Verificar o resultado gerado, no painel Description, conforme a figura seguinte:



Figura 4.142

A descrição apresentada para **PizzaPicante** (SpicyPizza) diz que todos os membros de **PizzaPicante** (SpicyPizza) são Pizzas e têm, pelo menos, um recheio com um tempero apimentado (spiciness) muito apimentado (Hot). Informa também que qualquer coisa que é uma Pizza e tem, pelo menos, um recheio com tempero apimentado (spiciness) muito apimentado (Hot) é um **PizzaPicante** (SpicyPizza).

No passo final do Exercício 41, criou-se uma restrição utilizando a expressão de classe **RecheioDePizza and temPicante some Forte**³⁸ ao invés de uma classe nomeada (Named class) com seu complemento da restrição (*filler*).

Esse complemento da restrição (*filler*) foi construído pela interseção entre a classe nomeada **RecheioDePizza** (PizzaTopping) e a restrição **temPicante some Forte**³⁹.

Uma outra forma para fazer esse procedimento é criar uma subclasse de **RecheioDePizza** (PizzaTopping) chamada **RecheioDePizzaPicante** (HotPizzaTopping) e defini-la com um recheio **Forte** (Hot | Picante) a partir da condição necessária **temPicante some Forte**⁴⁰. Pode-se então usar **temRecheio some RecheioDePizzaPicante**⁴¹ na definição de **PizzaPicante** (SpicyPizza).

Embora esta abordagem alternativa seja mais simples, é mais detalhada (*verbose*). A linguagem OWL permite encurtar as descrições e as definições das classes usando

³⁸ Em inglês: **PizzaTopping and hasSpiciness some Hot**

³⁹ Em inglês: **hasSpiciness some Hot**

⁴⁰ Em inglês: **hasSpiciness some Hot**

⁴¹ Em inglês: **hasTopping some HotPizzaTopping**

expressões de classe no lugar das classes nomeadas (Named class), como no exemplo acima.

Agora o MI pode ser ativado para determinar, na ontologia, quais pizzas são picantes (spicy).

Exercício 42: Usar o mecanismo de inferência (*Reasoner*) para classificar a ontologia

1º. Abrir a aba **Reasoner** (Mecanismo de inferência – MI), para lançar o processo de inferência, clicar em: **Start Reasoner**.

⚠ Verificar os resultados gerados, no painel de hierarquia inferida (*Inferred hierarchy*), conforme a figura seguinte:

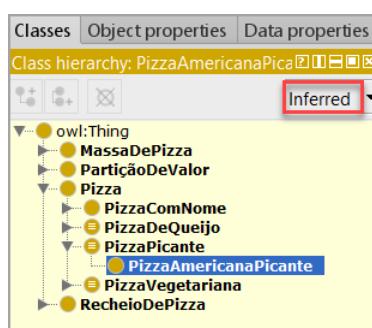


Figura 4.143

Depois de realizada a classificação automática, surge o painel de hierarquia inferida (*Inferred Hierarchy*). Observe-se que **PizzaAmericanaPicante** (AmericanHotPizza) foi classificada como subclasse de **PizzaPicante** (SpicyPizza): o MI computou automaticamente que qualquer indivíduo que é membro de **PizzaAmericanaPicante** (AmericanHotPizza) é também membro de **PizzaPicante** (SpicyPizza).

4.16. Restrições de cardinalidade (*Cardinality Restrictions*)

Em OWL, pode-se descrever a classe dos indivíduos que têm, pelo menos, um, ou *no máximo* ou *exatamente* um número específico de tipos de relação com outros indivíduos ou *Data Type values* (*valores de tipos de dados*). As restrições que descrevem essas classes são conhecidas como *Cardinality Restrictions* (*Restrições de Cardinalidade*).

Para uma dada propriedade P, uma *Minimum Cardinality Restriction* (*Restrição de Cardinalidade Mínima*) especifica o número mínimo de tipos de relação P dos quais um indivíduo deve participar.

Uma *Restrição Cardinalidade Máxima* (*Maximum Cardinality Restriction*) especifica o número máximo de tipos de relação P dos quais um indivíduo pode participar.

Uma *Cardinality Restriction* (*Restrição de Cardinalidade*) especifica o número exato de tipos de relação P dos quais um indivíduo participa.

Os tipos de relação (por exemplo, entre dois indivíduos) são considerados como tipos de relação separados quando se pode garantir que também são distintos os indivíduos que funcionam como complemento da restrição (*filler*) dos tipos de relação.

Por exemplo, a Figura 4.144 apresenta o indivíduo **Matthew** relacionado aos indivíduos **Nick** e **Hai**, através da propriedade **worksWith**. O indivíduo **Matthew** satisfaz uma restrição de *cardinalidade mínima* 2 na propriedade **worksWith**, caso os indivíduos **Nick** e **Hai** sejam indivíduos distintos.

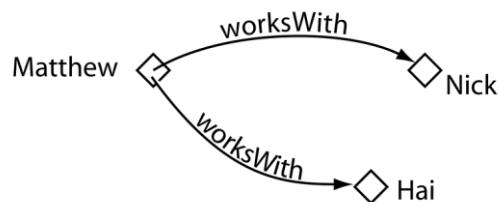


Figura 4.144 – Um exemplo de restrições de cardinalidade: Counting Relationships

Adiciona-se uma restrição de cardinalidade à ontologia de Pizza, criando uma subclasse de **Pizza** chamada **PizzalInteressante** (InterestingPizza), a qual será definida com três ou mais recheios.

Exercício 43: Criar uma classe PizzalInteressante (InterestingPizza) com pelo menos 3 recheios

- 1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **Pizza**.
- 2º. Criar uma classe **PizzalInteressante** (InterestingPizza), subclasse de **Pizza**.
- 3º. Selecionar a classe **PizzalInteressante** (InterestingPizza).
- 4º. Clicar no ícone **+ SubClassOf** (SubClasseDe), no painel **Classes Description** (Descrição de classes).
- 5º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição: **temRecheio min 3**

“**min 3**” é uma Restrição de cardinalidade mínima (Minimum Cardinality Restriction) de valor = 3.

Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:

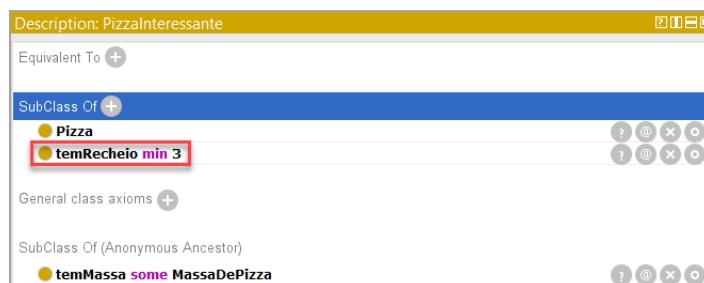


Figura 4.145

6º. Abrir a aba **Edit** e selecionar a função **Convert to defined class** (Converter em uma classe definida) (ver também a observação da Figura 4.141, p.95).

⚠ Verificar o resultado gerado, no painel **Description, conforme a figura seguinte:**



Figura 4.146

O que significa isso? A definição de **PizzalInteressante** (InterestingPizza) descreve o conjunto de indivíduos que são membros da classe **Pizza** e que têm pelo menos três relações do tipo **temRecheio** (hasTopping) com outros indivíduos (distintos um do outro).

Exercício 44: Usar o mecanismo de inferência (Reasoner**) para classificar a ontologia**

- 1º. Abrir a aba **Reasoner** (Mecanismo de inferência – MI), para lançar o processo de inferência, clicar em: **Start Reasoner**.
- 2º. Verificar os resultados gerados, no painel de **hierarquia inferida** (*Inferred hierarchy*), conforme a figura seguinte:

Após a classificação automática da ontologia, realizada pelo mecanismo de inferência (MI), aparece a janela de hierarquia inferida (*Inferred Hierarchy*). Após abertura da hierarquia, observe-se que **PizzalInteressante** (InterestingPizza) agora tem como subclasses **PizzaAmericana** (AmericanPizza), **PizzaAmericanaPicante** (AmericanHotPizza) e **PizzaSoho** (SohoPizza). Observe ainda que **PizzaMarguerita** (MargheritaPizza) não foi classificada sob **PizzalInteressante** (InterestingPizza) porque tem apenas dois tipos distintos de recheio.

4.17. Restrições de cardinalidade qualificadas (Qualified Cardinality Restrictions)⁴²

Na seção anterior, as restrições de cardinalidade foram descritas como restrições que especificam o número exato de relações de P em que um indivíduo deve participar.

⁴² Esse item consta na versão 1.3. (HORRIDGE; BRANDT, 2011, p. 75), não consta na versão 1.0 (HORRIDGE et al., 2004).

Nesta seção, trata-se das Qualified Cardinality Restrictions (Restrições de Cardinalidade Qualificada), que são mais específicas do que as restrições de cardinalidade, na medida em que elas declaram a classe de objetos dentro da restrição.

Pode-se adicionar uma restrição de cardinalidade qualificada à nossa ontologia de pizza.

Para fazer isso, basta criar uma subclasse de **PizzaComNome** (NamedPizza), chamada **PizzaQuatroQueijos** (FourCheesePizza), que será definida como tendo exatamente quatro recheios de queijo.

Exercício 45: Criar uma classe PizzaQuatroQueijos (FourCheesePizza) que tenha exatamente quatro recheios (diferentes) de queijo

- 1º. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaComNome** (NamedPizza), subclasse de **Pizza**.
- 2º. Criar uma classe **PizzaQuatroQueijos** (FourCheesePizza), subclasse de **Pizza**.
- 3º. Selecionar a classe **PizzaQuatroQueijos** (FourCheesePizza).
- 4º. Clicar no ícone  **SubClassOf** (SubClasseDe), no painel **Classes Description** (Descrição de classes).
- 5º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição: `temRecheio exactly 4 RecheioDeQueijo`

 Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:

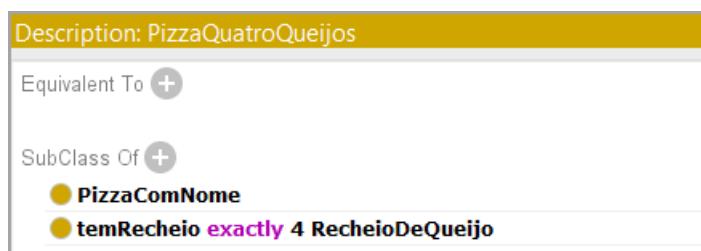


Figura 4.147

A definição de uma **PizzaQuattroQueijos** (FourCheesePizza) descreve o conjunto de indivíduos que são membros da classe **PizzaComNome** (NamedPizza) e que têm exatamente quatro relações de tipo **temRecheio** (hasTopping) com indivíduos da classe **RecheioDeQueijo** (CheeseTopping).

Com esta descrição, a classe **PizzaQuattroQueijos** (FourCheesePizza) ainda pode ter outras relações de tipo **temRecheio** (hasTopping) com outros tipos de recheios.

Para que possa ser especificado que a classe **PizzaQuattroQueijos** (FourCheesePizza) tenha quatro recheios de queijo e nenhum de outro tipo (de recheio), a palavra-chave "*only*" (quantificador universal) deve ser acrescentada na descrição. Isso significa que os únicos (*only*) tipos de recheio permitidos são recheio de queijo.



Uma restrição de cardinalidade não qualificada (Unqualified cardinality restriction) é exatamente idêntica a uma restrição de cardinalidade qualificada (Qualified cardinality restriction) com um complemento da restrição (*filler*): `owl:Thing`⁴³. Por exemplo, `hasTopping min 3` é o mesmo que `hasTopping min 3 owl:Thing`.

5. Propriedades de tipo de dados (*DataType Properties*)⁴⁴

Na Seção 4.4. (p. 30) do Capítulo 4, foram apresentadas as propriedades em OWL, mas foram apenas descritas as propriedades de objeto (*Object Properties*), isto é, as relações entre indivíduos.

Neste capítulo, serão apresentados e discutidos exemplos de propriedades do tipo de dados (*DataType Properties*). As propriedades do tipo de dados ligam um indivíduo a um valor de tipo de dados do esquema XML (XML Schema *DataType Value*) ou um literal RDF.

Em outras palavras, elas descrevem as relações entre um indivíduo e valores de dados. A maioria das propriedades descritas no Capítulo 4 não podem ser usadas com propriedades do tipo de dados (*DataType Properties*).

As características das propriedades que são aplicáveis às propriedades dos dados serão descritas mais adiante neste capítulo.

As propriedades do tipo de dados podem ser criadas usando o painel **Description**, após abertura das abas **Entities > Data Properties**, conforme mostrado na Figura 5.1:

⁴³ No Protégé 5, `owl:Thing` é inserido automaticamente (Figura 4.146, p. 92).

⁴⁴ Esse capítulo 5 consta na versão 1.3. (HORRIDGE; BRANDT, 2011, p. 76–82), não consta na versão 1.0 (HORRIDGE et al., 2004).

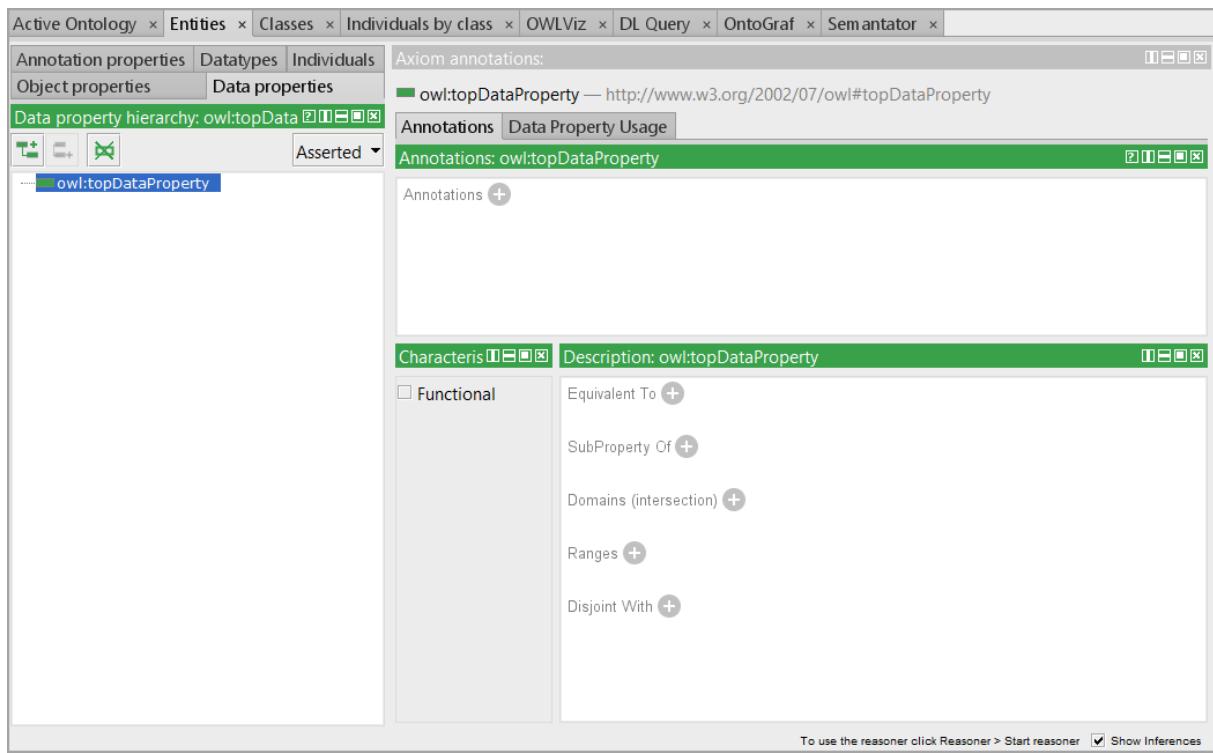


Figura 5.1 – Painel das Propriedades de dados (Data properties)

As propriedades de tipo de dados serão usadas para descrever o teor calórico das pizzas. Assim, alguns intervalos numéricos serão utilizados para classificar, de forma geral, pizzas específicas com valores altos ou baixos do teor calórico.

Para poder fazer isso, é preciso concluir as seguintes etapas:

- ◆ Criar uma propriedade de tipo de dados **temValorDoTeorCalórico** (**hasCalorificContentValue**), que será usada para indicar o conteúdo calórico de determinadas pizzas.
- ◆ Criar vários exemplos de indivíduos de pizza com conteúdo específico de calorias.
- ◆ Criar duas classes categorizando, de forma geral, as pizzas com valores altos ou baixos do teor calórico.

No Protégé 5, uma propriedade de tipo de dados pode ser usada para relacionar um indivíduo com um valor de dados concreto que pode ser digitado ou não.

Exercício 46: Criar uma propriedade do tipo de dados (*Data Type Properties*) denominada temValorDoTeorCalórico (*hasCalorificContentValue*)

1º. Abrir as abas **Entities** (Entidades) > **Data Properties** (Propriedades de dados), para criar uma propriedade de dados **temValorDoTeorCalórico** (**hasCalorificContentValue**), subclasse de **owl:topDataProperties**.

2º. Clicar em  , no painel **Data property hierarchy** (de cor verde), digitar na caixa de diálogo o nome da propriedade de dados: **temValorDoTeorCalórico**

⚠ Verificar o resultado gerado, conforme a figura seguinte:

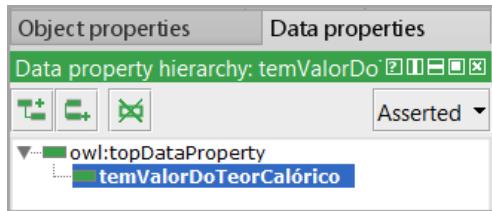


Figura 5.2

Não há nada intrínseco às propriedades de dados para evitar que seja realizada a classificação em nível de classe. Porém, serão criados alguns indivíduos de pizza como exemplos de classificação, com quantidades de calorias definidas de forma aleatória (motivos didáticos), pois, é difícil afirmar que todas as *Pizza Marguerita* têm 263 calorias.

Exercício 47: Criar exemplos de indivíduos de pizzas

- 1º. Abrir as abas **Entities** (Entidades) > **Individuals** (Indivíduos), para criar um indivíduo **ExemploMarguerita** (ExampleMargherita).
- 2º. Clicar em  , no painel **Individuals** (de cor roxa), digitar na caixa de diálogo o nome do indivíduo: **ExemploMarguerita**

 Verificar o resultado gerado, conforme a figura seguinte:

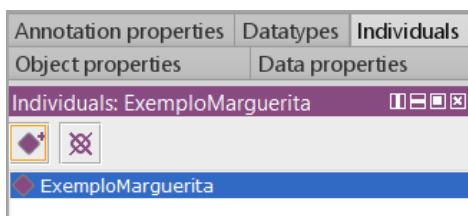


Figura 5.3

- 3º. Clicar no ícone  **Type** (Tipo), no painel **Individuals Description** (Descrição de indivíduos). Selecionar (ou verificar que está selecionado) o indivíduo **ExemploMarguerita** (Figura 5.3).
- 4º. Após abertura da caixa de diálogo, adicionar um tipo (*Type*) de **PizzaMarguerita** (MargheritaPizza). Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaMarguerita** (Figura 5.4). Clicar em: **OK**

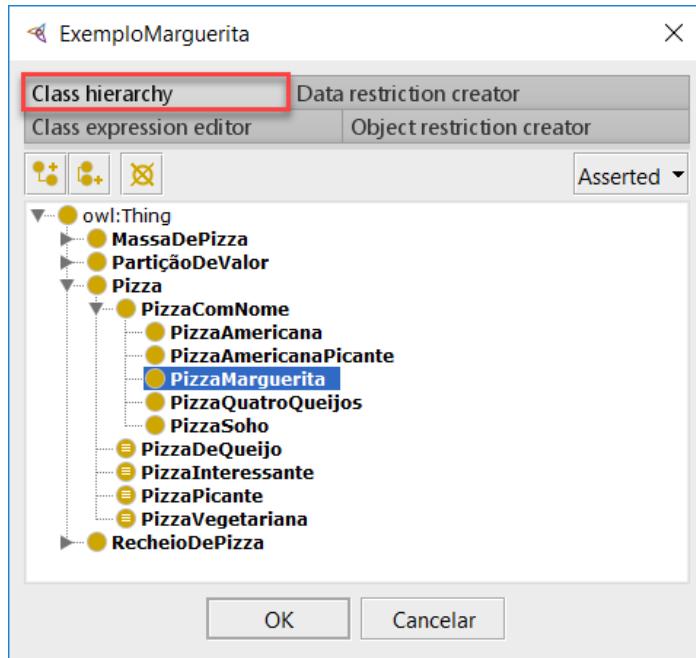


Figura 5.4

⚠ Verificar o resultado gerado, conforme a figura seguinte:

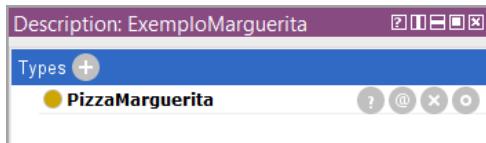


Figura 5.5

- 5º. Clicar no ícone + **Data property assertion** (Declaração de propriedade de dados), no painel **Property assertion** (Declaração de propriedade).
- 6º. Após abertura da caixa de diálogo, no painel **Data property**, selecionar a propriedade **temValorDoTeorCalórico** (`hasCalorificContentValue`) ①. Digitar a quantidade de calorias (value): **263** ② e selecionar o tipo de valor (*Data Type*), no menu suspenso: **xsd:integer** ③ (Figura 5.6).

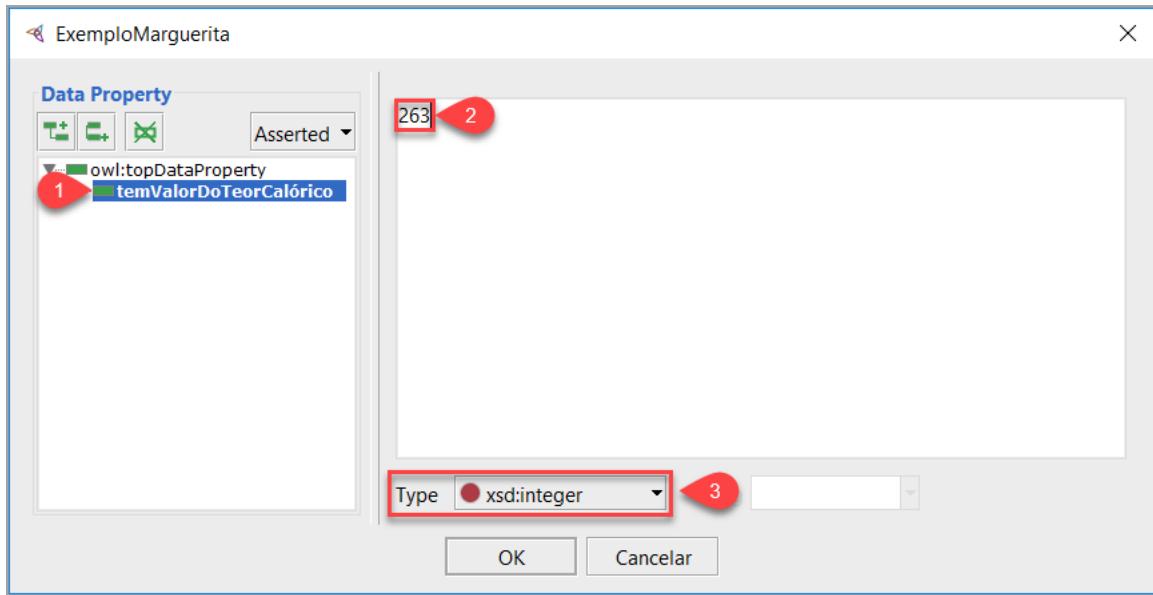


Figura 5.6

⚠ Verificar o resultado gerado, conforme a figura seguinte:



Figura 5.7

7º. **Repetir os passos anteriores** para criar vários indivíduos de pizza (por exemplo, as subclasses da classe PizzaComNome (NamedPizza)) com diferentes quantidades (aleatórias) de calorias (value):

- ◆ **ExemploAmericana** (ExempleAmerican): 478 calorias
- ◆ **ExemploAmericanaPicante** (ExempleAmericanHot): 327 calorias
- ◆ **ExemploQuatroQueijos** (ExempleFourCheese): 723 calorias.
- ◆ **ExemploSoho** (ExempleSoho): 515 calorias

⚠ Verificar o resultado gerado, conforme a figura seguinte:

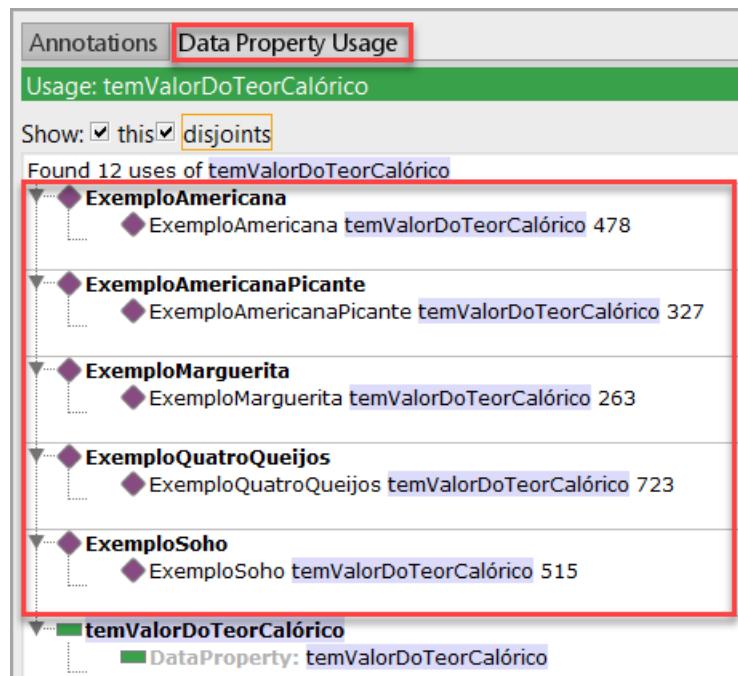


Figura 5.8

Uma propriedade de tipo de dados também pode ser usada em uma restrição para relacionar indivíduos a membros de um determinado tipo de dados. Os tipos de dados integrados (no Protégé 5) são especificados no vocabulário de esquema XML (XML Schema) e incluem inteiros (integers), pontos flutuantes (floats), cadeias de caracteres (strings), booleanos, etc.

Exercício 48: Criar uma restrição de tipo de dados (*Data Type*) para indicar que todas as pizzas têm um valor energético (expresso em calorias)

- 1º. Na aba **Classes > Class hierarchy** (Hierarquia de classe), selecionar a classe **Pizza**.
 - 2º. Clicar no ícone **+** **SubClassOf** (SubClasseDe), no painel **Classes Description** (Descrição de classes).
 - 3º. Após abertura da caixa de diálogo, abrir a aba **Data restriction creator** (Criador de restrição de dados) ① (Figura 5.9).
- 💡** **Data restriction creator** funciona da mesma maneira que **Object restriction creator** (Criador de restrição de objetos), mas o complemento da restrição (*filler*) é um tipo de dados nomeado (named *Data Type*), ou seja, integrados ao Protégé 5.
- 4º. Selecionar o tipo de restrição (Restriction type): “**some**” (existencial) ②
 - 5º. Selecionar a propriedade que receberá a restrição (Restricted property): **temValorDoTeorCalórico** (hasCalorificContentValue) ③
 - 6º. Selecionar o tipo de dados (*Data Type*) (Restriction *filler*): **xsd:integer** ④. Clicar em: **OK**

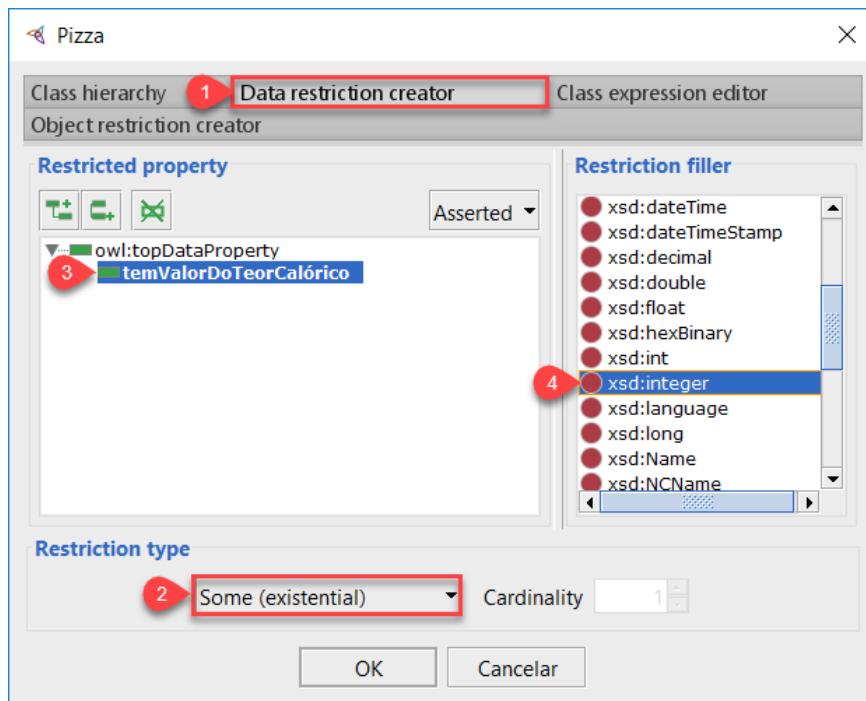


Figura 5.9

⚠ Verificar o resultado gerado, no painel Description, conforme a figura seguinte:



Figura 5.10



Agora todas as pizzas foram declaradas com pelo menos um teor calórico (e esse valor deve ser um número inteiro: **integer**).

Além de usar o conjunto predefinido de tipos de dados (*Restriction filler*), é possível também caracterizar o uso de um tipo de dados especificando restrições sobre os valores possíveis. Por exemplo, pode-se definir um intervalo para um valor numérico.

Usa-se a propriedade de tipo de dados, gerada anteriormente, para criar classes definidas que especificam um intervalo de valores pertinentes. Serão criadas definições que usam as facetas (*facets*) *minInclusive* e *maxExclusive*, aplicáveis em tipos de dados numéricos. A classe **PizzaDeTeorCalóricoElevado** (HighCaloriePizza) será definida como “qualquer pizza que tenha um poder calórico superior ou igual a 400”.

Exercício 49: Criar uma classe PizzaDeTeorCalóricoElevado (HighCaloriePizza) que tenha um valor energético superior ou igual a 400 calorias

1º. Na aba **Classes > Class hierarchy** (Hierarquia de classe), selecionar a classe de **Pizza**.

2º. Criar uma classe  **PizzaDeTeorCalóricoElevado** (HighCaloriePizza), subclasse de **Pizza**.

3º. Clicar no ícone  **SubClassOf** (SubClasseDe), no painel **Classes Description** (Descrição de classes).

4º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição⁴⁵: `temValorDoTeorCalórico some xsd:integer [>= 400]` (isto é: “qualquer pizza que tenha um poder calórico superior ou igual a 400”). Clicar em: **OK**

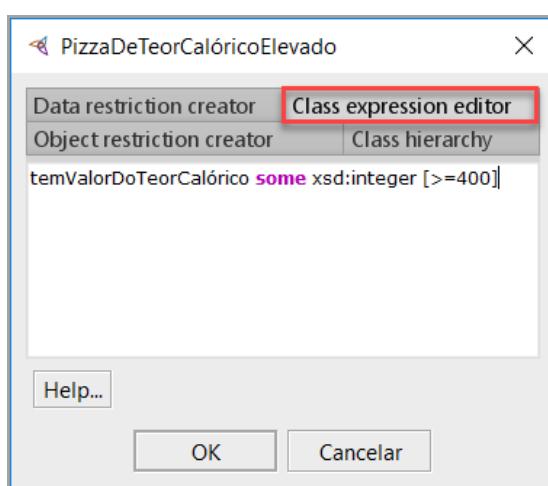


Figura 5.11

 Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:

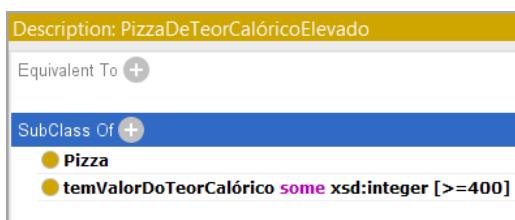


Figura 5.12

5º. Na aba **Classes > Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaDeTeorCalóricoElevado** (HighCaloriePizza).

6º. Abrir a aba **Edit** e selecionar a função **Convert to defined class** (Converter em uma classe definida).

7º. A classe **PizzaDeTeorCalóricoElevado** (HighCaloriePizza) convertida em classe definida (defined class) , conforme a figura seguinte:

Description: PizzaDeTeorCalóricoElevado

Equivalent To +
● Pizza and (temValorDoTeorCalórico some xsd:integer[>= 400])

SubClass Of +

Figura 5.13

- 8º. **Repetir os passos anteriores** para criar a classe **PizzaDeTeorCalóricoBaixo** (LowCaloriePizza) e digitar a restrição: temValorDoTeorCalórico **some** xsd:integer[< 400] (isto é: “qualquer pizza que tenha um teor calórico menor a 400”). Clicar em: **OK**

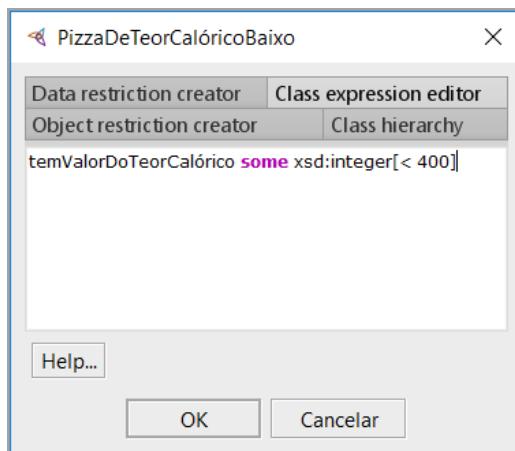


Figura 5.14

💡 Observe-se que a definição de PizzaDeTeorCalóricoBaixo (LowCaloriePizza) [<400] não se sobrepõe a PizzaDeTeorCalóricoElevado (HighCaloriePizza) [>=400].

⚠ Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:

Description: PizzaDeTeorCalóricoBaixo

Equivalent To +
● Pizza and (temValorDoTeorCalórico some xsd:integer[< 400])

SubClass Of +

Figura 5.15

Duas categorias de pizza foram criadas para abranger qualquer indivíduo que tenha seu teor calórico especificado. Agora é preciso verificar, com o mecanismo de inferências (MI), se a classificação é válida para os exemplos de indivíduos (instâncias) que foram definidos.

Exercício 50: Classificar os indivíduos de pizzas de acordo com os critérios da propriedade temValorDoTeorCalórico (hasCalorificContentValue)

- 1º. Abrir a aba **Reasoner** (Mecanismo de inferência – MI), para lançar o processo de inferência, clicar em: **Start Reasoner**.

2º. Na aba **Classes > Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaDeTeorCalóricoElevado** (HighCaloriePizza).

⚠ Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:

The screenshot shows the Description panel for the class **PizzaDeTeorCalóricoElevado**. The panel is divided into several sections:

- Equivalent To**: Shows the class definition: **Pizza and (temValorDoTeorCalórico some xsd:integer[>= 400])**.
- SubClass Of**: Shows the super-class **Pizza**.
- General class axioms**: Shows two axioms:
 - temValorDoTeorCalórico some xsd:integer**
 - temMassa some MassaDePizza**
- Instances**: Shows three instances:
 - ExemploAmericana**
 - ExemploQuatroQueijos**
 - ExemploSoho**

A red box highlights the **Instances** section.

Figura 5.16

→ Lembrando os indivíduos (Instances), teor calórico [≥ 400] (Exercício 47, p. 102):

- ♦ ExemploAmericana (ExempleAmerican): 478 calorias
- ♦ ExemploQuatroQueijos (ExempleFourCheese): 723 calorias
- ♦ ExemploSoho (ExempleSoho): 515 calorias

3º. Na aba **Classes > Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaDeTeorCalóricoBaixo** (LowCaloriePizza).

⚠ Verificar o resultado gerado, no painel **Description**, conforme a figura seguinte:

The screenshot shows the Protege ontology editor interface. At the top, it says "Description: PizzaDeTeorCalóricoBaixo". Below that, under "Equivalent To", there is an axiom: "Pizza and (temValorDoTeorCalórico some xsd:integer[< 400])". Under "SubClass Of", it lists "Pizza". Under "General class axioms", there are two axioms: "temValorDoTeorCalórico some xsd:integer" and "temMassa some MassaDePizza". The "Instances" section is highlighted with a red border and contains two entries: "ExemploAmericanaPicante" and "ExemploMarguerita". At the bottom right, there are buttons for "Reasoner active" and "Show Inferences".

Figura 5.17

- Lembrando os indivíduos (Instances), teor calórico [< 400] (Exercício 47, p. 102):
- ◆ ExemploAmericanaPicante (ExempleAmericanHot): 327 calorias
 - ◆ ExemploMarguerita (ExampleMargherita): 263 calorias

Finalmente, quantos valores de calorias diferentes podem ser assumidos por um indivíduo pizza? Sem dúvida, a resposta é apenas uma. Lembrando que existe um critério de propriedade que estipula que uma propriedade desse tipo pode ser assumida por apenas um indivíduo ao mesmo tempo.

Ao definir uma propriedade de objeto como *funcional* (Seção 4.6.1., p. 37), afirma-se que qualquer indivíduo pode estar relacionado, no máximo, a um outro indivíduo por meio dessa propriedade. A característica funcional é também disponível para as propriedades dos dados. (É atualmente a única característica que é possível usar nas propriedades dos dados). Ao tornar a propriedade **temValorDoTeorCalórico** (`hasCalorificContentValue`) funcional, significa que qualquer pizza só pode ter, no máximo, um (único) valor de calorias.

Exercício 51: Tornar funcional a propriedade do tipo de dados (*Data Type Properties*) **temValorDoTeorCalórico (`hasCalorificContentValue`)**

- 1º. Abrir as abas **Entities** (Entidades) > **Data Properties** (Propriedades de dados), e selecionar a propriedade de dados **temValorDoTeorCalórico** (`hasCalorificContentValue`).
 - 2º. No painel **Characteristics**, selecionar a única opção: **Funcional**
- ⚠ Verificar o resultado gerado, conforme a figura seguinte:**

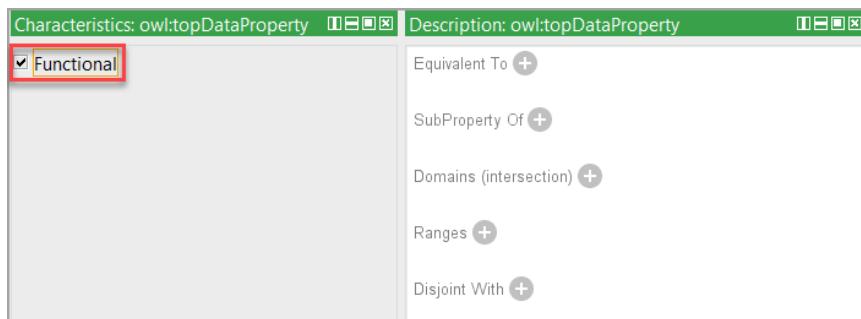


Figura 5.18

- 3º. Para testar o funcionamento da ontologia com a característica “funcional” da propriedade do tipo de dados **temValorDoTeorCalórico** (`hasCalorificContentValue`), adicionar um valor de calorias: 123, conforme a figura seguinte:



Figura 5.19

- 4º. Abrir a aba **Reasoner** (Mecanismo de inferência – MI), para lançar o processo de inferência, clicar em: **Start Reasoner**.

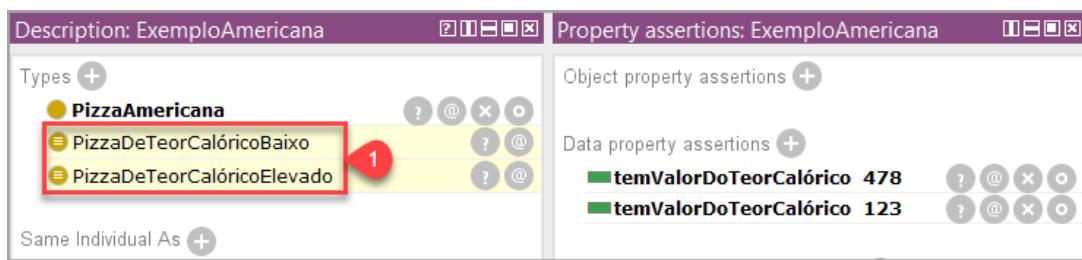
⚠️ Verificar o resultado gerado, conforme a figura seguinte:



Figura 5.20

💡 Conforme previsto, a adição do valor de calorias ①, gerou uma inconsistência da ontologia ②.

- 5º. Removendo a restrição “Funcional” da propriedade de dados **temValorDoTeorCalórico** (`hasCalorificContentValue`) e ativando novamente o MI, no resultado da inferência, constam os dois valores ① (baixo e elevado), incoerentes, para a mesma pizza:



6º. Restabelecer as configurações iniciais: deletar o segundo valor (123) e marcar novamente a característica “Funcional”, da propriedade de dados **temValorDoTeorCalórico** (hasCalorificContentValue). Verificar o resultado.

6. Mais sobre o raciocínio de mundo aberto (Open World Reasoning – OWR)

Os exemplos dessa seção demonstram novas nuances do Raciocínio de Mundo Aberto.

Uma classe **PizzaNãoVegetariana** (NonVegetarianPizza) será criada para complementar a categorização de pizzas da classe **PizzaVegetariana** (VegetarianPizza). A NonVegetarianPizza deve conter todas as pizzas que não pertencem à classe **PizzaVegetariana** (VegetarianPizza). Para isso, uma classe que é o complemento de **PizzaVegetariana** (VegetarianPizza) será criada.

Uma classe complemento (complement class) contém todos os indivíduos que não estão contidos na classe da qual ela é complemento. Portanto, ao criar **PizzaNãoVegetariana** (NonVegetarianPizza) (subclasse de **Pizza**) como complemento de **PizzaVegetariana** (VegetarianPizza), ela deve conter todas as pizzas que não são membros de **PizzaVegetariana** (VegetarianPizza).

Exercício 52: Criar uma classe **PizzaNãoVegetariana** (NonVegetarianPizza), subclasse da classe **Pizza** e disjunta da classe **PizzaVegetariana** (VegetarianPizza)

1º. Na aba **Classes > Class hierarchy** (Hierarquia de classe), selecionar a classe **Pizza**.

2º. Criar uma classe  **PizzaNãoVegetariana** (NonVegetarianPizza), subclasse de **Pizza**.

3º. Selecionar **PizzaNãoVegetariana** (NonVegetarianPizza).

4º. Clicar no ícone  **Disjoint With** (Tornar disjunto de), no painel **Description** (Descrição).

 Aparece uma caixa de diálogo.

5º. Verificar que a aba **Class hierarchy** (Hierarquia de classe) está aberta. Selecionar a classe **PizzaVegetariana** (VegetarianPizza), que deve ser disjunta da classe **PizzaNãoVegetariana** (NonVegetarianPizza). Clicar em: **OK**

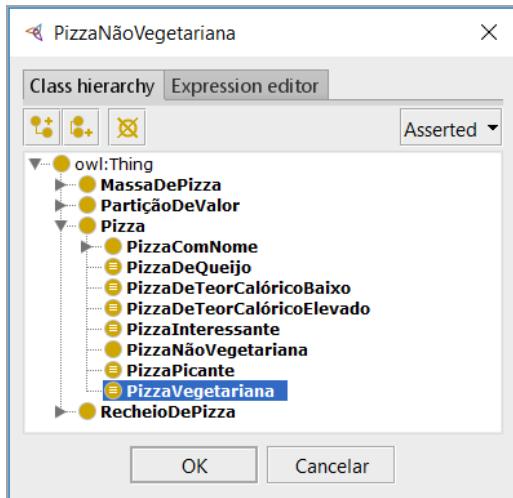


Figura 6.1

⚠ Verificar os resultados no painel Description:



Figura 6.2

Deseja-se definir uma **PizzaNãoVegetariana** (NonVegetarianPizza) como uma **Pizza** que não é **PizzaVegetariana** (VegetarianPizza).

Exercício 53: Tornar complemento da classe PizzaVegetariana (VegetarianPizza) a classe PizzaNãoVegetariana (NonVegetarianPizza)

- 6º. Na aba **Classes > Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaNãoVegetariana** (NonVegetarianPizza).
- 7º. Clicar no ícone **+ Subclass Of** (Subclasse de), no painel **Description** (Descrição).
- 8º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição: **Pizza and not PizzaVegetariana**

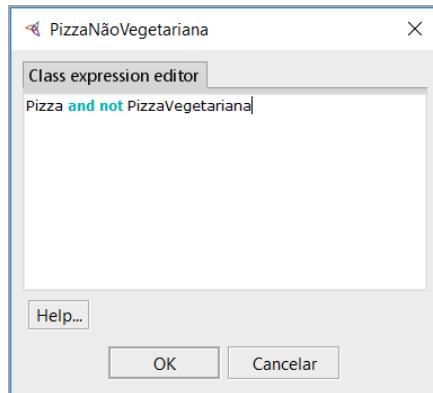


Figura 6.3

9º. Após digitar a restrição completa (com a sintaxe correta), conforme a figura seguinte, clicar em **OK**:

⚠ Verificar os resultados no painel Description:



Figura 6.4

Remover a ocorrência Pizza (isolada).

No entanto, é preciso adicionar a classe **Pizza** às *condições necessárias e suficientes*, uma vez que, no momento, a definição da classe **PizzaNãoVegetariana** (NonVegetarianPizza) diz que um indivíduo que não é um membro da classe **PizzaVegetariana** (VegetarianPizza) (ou qualquer outra coisa) é uma **PizzaNãoVegetariana** (NonVegetarianPizza).

Exercício 54: Adicionar a classe Pizza às condições necessárias e suficientes da classe PizzaNãoVegetariana (NonVegetarianPizza)

1º. Na aba **Classes > Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaNãoVegetariana** (NonVegetarianPizza).

2º. Abrir a aba **Edit** e selecionar a função **Convert to defined class** (Converter em uma classe definida):

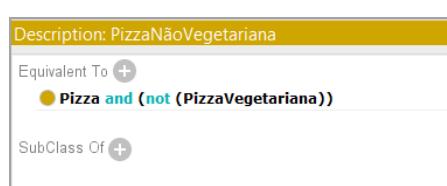


Figura 6.5

O complemento de uma classe inclui todos os indivíduos que não são membros da classe. Ao tornar **PizzaNãoVegetariana** (NonVegetarianPizza) uma subclasse de **Pizza** e complemento de **PizzaVegetariana** (VegetarianPizza), indica-se que os indivíduos que são Pizzas e não são membros de **PizzaVegetariana** (VegetarianPizza) são membros de **PizzaNãoVegetariana** (NonVegetarianPizza). Observe-se que **PizzaVegetariana** (VegetarianPizza) e **PizzaNãoVegetariana** (NonVegetarianPizza) são disjuntas: se um indivíduo é membro de VegetarianPizza ele não pode ser um membro de **PizzaNãoVegetariana** (NonVegetarianPizza).

Exercício 55: Usar o mecanismo de inferência (*Reasoner*) para computar as classes PizzaNãoVegetariana e PizzaVegetariana

- 1º. Abrir a aba **Reasoner** (Mecanismo de inferência – MI), para lançar o processo de inferência, clicar em: **Start Reasoner**.

⚠ Verificar o resultado gerado, conforme a figura seguinte:

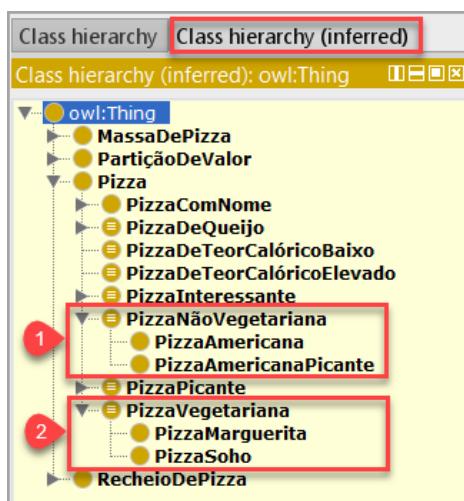


Figura 6.6

A hierarquia de classes inferidas deve parecer com a Figura 6.6. As classes **PizzaMarguerita** (MargheritaPizza) e **PizzaSoho** (SohoPizza) foram classificadas como subclasses de **PizzaVegetariana** (VegetarianPizza) ②; as classes **PizzaAmericana** (AmericanPizza) e **PizzaAmericanaPicante** (AmericanHotPizza) foram classificadas como **PizzaNãoVegetariana** (NonVegetarianPizza) ①. As coisas pareciam estar corretas... Entretanto, será adicionado uma pizza sem axioma de fechamento (closure axiom) à propriedade **temRecheio** (hasTopping).

Exercício 56: Criar uma classe PizzaSemFechamento (UnclosedPizza), subclasse de PizzaComNome (NamedPizza), com um recheio de Muçarela (Mozzarella)

- 1º. Na aba **Classes > Class hierarchy** (Hierarquia de classe), selecionar a classe **PizzaComNome** (NamedPizza).

2º. Criar uma classe  **PizzaSemFechamento** (UnclosedPizza), subclasse de **PizzaComNome** (NamedPizza).

3º. Clicar no ícone  **Subclass Of** (Subclasse de), no painel **Description** (Descrição).

4º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição: **temRecheio some RecheioDeMuçarela**

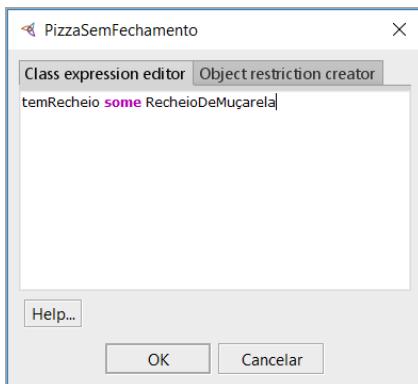


Figura 6.7

5º. Após digitar a restrição completa (com a sintaxe correta), conforme a figura seguinte, clicar em **OK**:

 **Verificar os resultados no painel Description:**

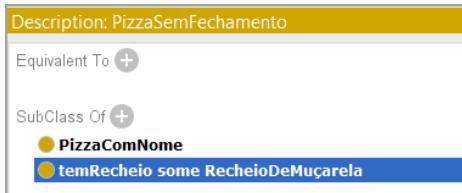


Figura 6.8

Se um indivíduo é um membro de **PizzaSemFechamento** (UnclosedPizza), é necessário que ele seja uma **PizzaComNome** (NamedPizza), e que tenha pelo menos uma relação de tipo **temRecheio** (hasTopping) com um indivíduo que é um membro da classe **RecheioDeMuçarela** (MozzarellaTopping).

Por causa da Suposição de Mundo Aberto (*Open World Assumption*) e do fato de que não se adicionou um axioma de fechamento (Closure axiom) à propriedade **temRecheio** (hasTopping), uma **PizzaSemFechamento** (UnclosedPizza) pode ter recheios adicionais que não são tipos de **RecheioDeMuçarela** (MozzarellaTopping).

Exercício 57: Usar o mecanismo de inferência (Reasoner) para classificar a ontologia

1º. Abrir a aba **Reasoner** (Mecanismo de inferência – MI), para lançar o processo de inferência, clicar em: **Start Reasoner**.

⚠ Verificar o resultado gerado, conforme a figura seguinte:

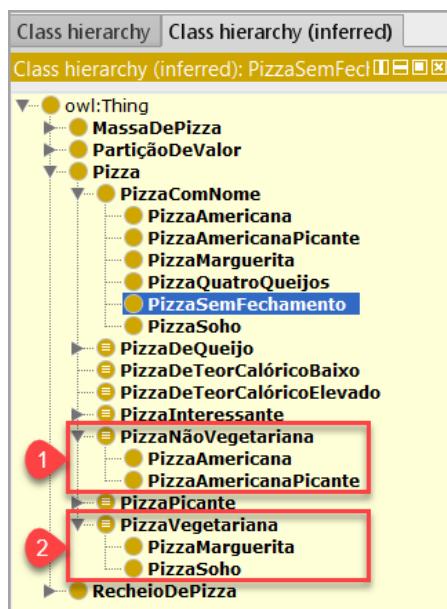


Figura 6.9

Observe-se, na hierarquia de classes inferidas (Figura 6.9), que **PizzaSemFechamento** (UnclosedPizza) não é uma **PizzaVegetariana** (VegetarianPizza) nem uma **PizzaNãoVegetariana** (NonVegetarianPizza).

A **PizzaSemFechamento** (UnclosedPizza) não foi classificada como **PizzaVegetariana** (VegetarianPizza), em função do Raciocínio de mundo aberto (OWR). O MI não pode determinar se **PizzaSemFechamento** (UnclosedPizza) é uma **PizzaVegetariana** (VegetarianPizza) porque não existe um axioma de fechamento em **temRecheio** (hasTopping), e a **Pizza** pode ter outros recheios. **PizzaSemFechamento** (UnclosedPizza) poderia ser classificada como **PizzaNãoVegetariana** (NonVegetarianPizza), desde que não fosse classificada como **PizzaVegetariana** (VegetarianPizza).

No entanto, o raciocínio de mundo aberto não demanda que a classe **PizzaSemFechamento** (UnclosedPizza), por não ser uma **PizzaVegetariana** (VegetarianPizza), seja uma **PizzaVegetariana** (VegetarianPizza): ela pode ser **PizzaVegetariana** (VegetarianPizza) e pode não ser **PizzaVegetariana** (VegetarianPizza). Dessa forma, **PizzaSemFechamento** (UnclosedPizza) não pode ser classificada como **PizzaNãoVegetariana** (NonVegetarianPizza).

7. Criar outras estruturas OWL no Protégé

Nessa seção discute-se como criar outras estruturas OWL usando o Protégé 5. Tais construções não são parte principal do tutorial e podem ser criadas em um novo projeto. No entanto, é melhor seguir este capítulo usando a ontologia da pizza já construída.

7.1. Criar indivíduos (*Individuals*)

O OWL permite definir indivíduos e declarar propriedades sobre eles. Os indivíduos podem também ser usados em descrições de classe, a saber, em restrições `hasValue` (temValor) e Classes enumeradas (*Enumerated classes*).

Para criar indivíduos no Protégé 5, usa-se a guia **Individuals** (Indivíduos).

Objetiva-se descrever o país de origem dos recheios usados nas Pizzas. Em primeiro lugar, é preciso adicionar vários países à ontologia de Pizza. Os países, por exemplo, England (Inglaterra), Italy (Itália), America (Estados Unidos), são geralmente considerados como indivíduos (seria incorreto ter uma classe **Inglaterra**, por exemplo, pois, seus membros seriam considerados "coisas que são instâncias da Inglaterra").

Para criar essa situação na ontologia de Pizza, será criada uma classe **Country** (País) e então será povoada com indivíduos.

Exercício 58: Criar uma classe chamada País (Country) e povoar esta classe de alguns indivíduos

- ◆ Na aba **Classes > Class hierarchy** (Hierarquia de classe), selecionar a classe **owl:Thing**.
- ◆ Criar uma classe  País (Country), subclasse de **owl:Thing**.
- ◆ Abrir as abas **Entities** (Entidades) > **Individuals** (Indivíduos), para criar os indivíduos (ou instâncias).
- ◆ Lembre-se de que “indivíduo” é outro nome para “instância” na terminologia de ontologia.
- ◆ Clicar em  , no painel **Individuals** (de cor roxa), digitar na caixa de diálogo o nome do indivíduo: **Itália** (Italy).
- ◆ Clicar no ícone  **Type** (Tipo), no painel **Individuals Description** (Descrição de indivíduos). Selecionar (ou verificar que está selecionado) o indivíduo **Itália**.
- ◆ Após abertura da caixa de diálogo, adicionar um tipo (Type) de **País**. Na aba **Class hierarchy** (Hierarquia de classe), selecionar a classe **País** (Country). Clicar em: **OK**
- ◆ Verificar o resultado gerado, conforme a figura seguinte:



Figura 7.1

- ◆ A seleção da classe País (Country), na hierarquia de classes, torna a Itália (Italy) um indivíduo da classe País (Country).
- ◆ **Repetir os passos anteriores** para criar vários indivíduos da classe País (Country):
 - ◆ Estados Unidos (America)
 - ◆ Inglaterra (England)
 - ◆ França (France)
 - ◆ Alemanha (Germany)

⚠ Verificar o resultado gerado, na aba Classes > Class hierarchy (Hierarquia de classe), no painel Description, conforme a figura seguinte:



Figura 7.2

Lembrando que OWL não usa o *Unique Name Assumption* (Postulado de nomes únicos) (Seção 3.2.1., p. 14). Por isso, os indivíduos podem ser declarados como `owl:sameAs` (igualA) ou `owl:differentFrom` (diferenteDe) de outros indivíduos.

No Protégé 5, essas afirmações podem ser realizadas usando as seções **SameAs** e **differentFrom**, do painel **Individuals Description** (Descrição de indivíduos), conforme a figura seguinte:

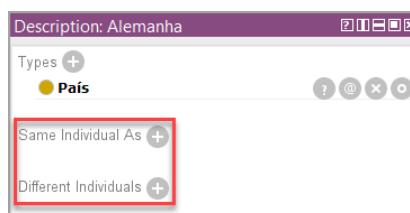


Figura 7.3

Tendo criado alguns indivíduos pode-se usá-los em descrições de classes, conforme descrito nas seções 7.2. e 7.3. , seguintes.

7.2. Restrição temValor (hasValue)

Uma restrição **hasValue** (temValor), representada pelo símbolo \exists , descreve o conjunto de indivíduos que possui *pelo menos um* tipo de relação através da propriedade com indivíduo específico (specific individual).

Por exemplo, a restrição **hasValue** (temValor), representada pela declaração `temPaísDeOrigem value Itália46` (onde **Itália** é um indivíduo), descreve o conjunto de indivíduos (a classe anônima de indivíduos) que têm, pelo menos, um tipo de relação através da propriedade **hasCountryOfOrigin** (temPaísDeOrigem) com o indivíduo específico **Italy** (Itália).

Objetiva-se especificar a origem dos ingredientes na ontologia de Pizza. Por exemplo, pode-se dizer que **MozzarellaTopping** (RecheioDeMuçarela) vem da **Italy** (Itália). Alguns países foram inseridos na ontologia de Pizza, os quais são representados como indivíduos. Pode-se usar a restrição **hasValue** (temValor) junto a esses indivíduos para especificar que o país de origem de **MozzarellaTopping** (RecheioDeMuçarela) é **Italy** (Itália).

Exercício 59: Criar uma restrição temValor (hasValue) para especificar que RecheioDeMuçarela (MozzarellaTopping) tem Itália (Italy) como país de origem.

1º. Abrir as abas **Entities** (Entidades) > **Object Properties** (Propriedades de objeto), para criar uma propriedade de objeto **temPaísDeOrigem** (`hasCountryOfOrigin`), subclasse de `owl:topObjectProperties`.

2º. Clicar em  no painel **Object property hierarchy** (de cor azul), digitar na caixa de diálogo o nome da propriedade de dados: `temPaísDeOrigem`

 Verificar o resultado gerado, conforme a figura seguinte:

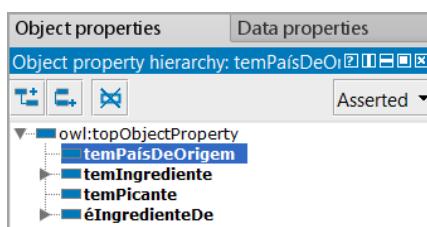


Figura 7.4

3º. Na aba **Classes > Class hierarchy** (Hierarquia de classe), selecionar a classe de **RecheioDeMuçarela** (MozzarellaTopping).

4º. Clicar no ícone  **SubClassOf** (SubClasseDe), no painel **Classes Description** (Descrição de classes).

⁴⁶ Em inglês: `hasCountryOfOrigin value Italy (ou hasCountryOfOrigin \exists Italy)`

5º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição: `temPaísDeOrigem value Itália`
Clicar em: **OK**

⚠ Verificar o resultado gerado, conforme a figura seguinte:



Figura 7.5

As condições definidas (Figura 7.5) para `RecheioDeMuçarela` (MozzarellaTopping) informam que: os indivíduos membros da classe `RecheioDeMuçarela` (MozzarellaTopping) são também membros da classe `RecheioDeQueijo` (CheeseTopping) e estão relacionados ao indivíduo `Itália` (Italy), pela propriedade `temPaísDeOrigem` (hasCountryOfOrigin), e estão relacionados a pelo menos um membro da classe `Moderado` (Mild) pela propriedade `temPicante` (hasSpiciness).

Em linguagem natural, pode-se dizer que coisas que são tipos de `RecheioDeMuçarela` (MozzarellaTopping) são também `RecheioDeQueijo` (CheeseTopping), vêm da `Itália` (Italy) e têm `temPicante` (hasSpiciness) `Moderado` (Mild) (moderadamente apimentados).

⚠ Com os mecanismos de inferência (MI) atuais, a classificação automática não é completa para os indivíduos. Usa-se indivíduos em descrições de classe com cuidado, pois resultados inesperados podem ser gerados pelo MI.

7.3. Classes enumeradas (*Enumerated classes*)⁴⁷

Além de descrever as classes através de (sub)classes⁴⁸ nomeadas e (sub)classes anônimas, como as restrições, a linguagem OWL permite definir classes especificando exatamente os indivíduos que são seus membros.

Por exemplo, pode-se definir uma classe `DaysOfTheWeek` (DiasDaSemana) como a classe que contém os indivíduos (e somente os indivíduos): `Sunday` (Domingo), `Monday` (Segunda), `Tuesday` (Terça), `Wednesday` (Quarta), `Thursday` (Quinta), `Friday` (Sexta) e `Saturday` (Sábado). Classes definidas dessa forma são conhecidas como *Enumerated classes* (Classes enumeradas).

No Protégé 5, as classes enumeradas são definidas usando-se o **Class expression editor** (editor de expressões de classe). Os indivíduos que compõem a classe enumerada são

⁴⁷ <https://www.w3.org/TR/owl-ref/#EnumeratedClass>

⁴⁸ Chamadas “superclasses” nas versões anteriores à versão atual de Protégé (v. 5).

listados, separados por vírgulas e dentro de chaves, por exemplo: {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday}.

Os indivíduos já devem ter sido criados na ontologia. As classes enumeradas descritas dessa forma são classes anônimas: são as classes dos indivíduos e apenas dos indivíduos listados na enumeração. No Protégé 5, pode-se anexar esses indivíduos a uma classe nomeada (Named Class), criando a enumeração como uma classe equivalente (Equivalent class).

Exercício 60: Converter a classe País (Country) em uma classe enumerada (Enumerated classes)

- 1º. Na aba **Classes > Class hierarchy** (Hierarquia de classe), selecionar a classe País (Country).
- 2º. Clicar no ícone **Equivalent To** (Equivalente a), no painel **Classes Description** (Descrição de classes).
- 3º. Após abertura da caixa de diálogo, abrir a aba **Class expression editor** (Editor de expressão de classe) e digitar a restrição: {Alemanha, Estados Unidos, França, Inglaterra, Itália} Clicar em: **OK**

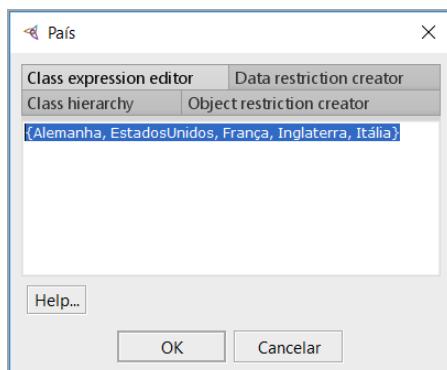


Figura 7.6

Verificar o resultado gerado, conforme a figura seguinte:



Figura 7.7

Um indivíduo que é um membro da classe **País** (Country) é na verdade um dos indivíduos listados (ou seja, um daqueles da lista {Alemanha, Estados Unidos, França, Inglaterra, Itália}). Formalmente, a classe **País** (Country) contém os mesmos indivíduos que a classe anônima e é equivalente a essa classe anônima, definida pela enumeração, conforme apresentado na seguinte:

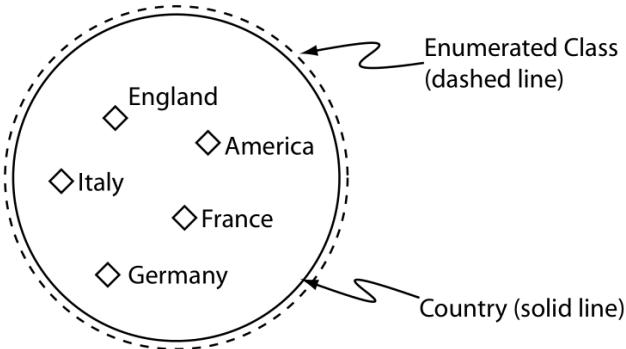


Figura 7.8 - Esquema da classe *Country* como equivalente a uma classe enumerada

⚠️ Obviamente, isto não é uma lista completa de países, mas para os propósitos desta ontologia (e deste exemplo), atende às necessidades didáticas.

7.4. Propriedades de Anotação (Annotation Properties)⁴⁹

A linguagem OWL permite que as classes, as propriedades, os indivíduos e o próprio cabeçalho da ontologia sejam comentados usando metadados. Esses metadados podem assumir a forma de informações de auditoria ou de informações editoriais. Por exemplo: comentários, data de criação, autor, referências para pesquisas tais como páginas web, etc.

O OWL-Full não impõe quaisquer restrições a essas propriedades, mas OWL-DL tem restrições ao uso da anotação, sendo que as duas mais importantes são:

- 1) O complemento da restrição (*filler*) para as propriedades de anotação deve ter um dado literal, uma URI de referência ou um indivíduo. Um dado literal é um caractere de representação de um *DataType value* (valor de tipo de dados), por exemplo, "Matthew", "25", "3.11".
- 2) As propriedades de anotação não podem ser usadas em axiomas que atuam sobre propriedades. Por exemplo, elas não podem ser usadas na hierarquia de propriedades, de forma que não podem ter subpropriedades, ou ser subpropriedade de outra propriedade. Também não podem ter Domínio (Domain) e Escopo (Range) específico.

OWL tem cinco propriedades de anotação predefinidas que podem ser usadas para fazer comentários em classes (inclusive classes anônimas, tais como restrições), propriedades e indivíduos:

⁴⁹ <https://www.w3.org/TR/owl-ref/#Annotations>

- 1) **owl:versionInfo** — Em geral, o escopo (range) dessa propriedade é uma cadeia de caracteres (string).
- 2) **rdfs:label** — O escopo (range) é uma cadeia de caracteres (string). Esta propriedade é usada para adicionar nomes significativos (para humanos) aos elementos da ontologia, tais como classes, propriedades e indivíduos. **rdfs:label** é também usado para fornecer nomes multilíngues para elementos de ontologia.
- 3) **rdfs:comment** — O escopo (range) é uma cadeia de caracteres (string).
- 4) **rdfs:seeAlso** — O escopo (range) é uma URI usada para identificar recursos conexos.
- 5) **rdfs:isDefinedBy** — o escopo (range) é uma URI usada para referenciar uma ontologia que define elementos da ontologia, tais como classes, propriedades e indivíduos.

Por exemplo, a propriedade de anotação **rdfs:comment** é usada para armazenar comentários sobre as classes no Protégé 5. A propriedade de anotação **rdfs:label** pode ser usada para fornecer nomes alternativos para classes, propriedades, etc.

Existem também propriedades de anotação que podem ser usadas para comentar uma ontologia. As propriedades de anotação de ontologias (listadas abaixo) têm como escopo (range) uma URI, usada para fazer referência a outra ontologia. É também possível usar a propriedade de anotação **owl:VersionInfo** para comentários de ontologia.

- 1) **owl:priorVersion** identifica versões anteriores da ontologia.
- 2) **owl:backwardsCompatibleWith** identifica versão anterior da ontologia que é compatível com a versão atual. Isso quer dizer que todos os identificadores da versão anterior possuem o mesmo significado na versão atual. Assim, todas as ontologias ou aplicações que se referem à versão anterior podem mudar com segurança para a nova versão.
- 3) **owl:incompatibleWith** identifica a versão anterior de uma ontologia que não é compatível com a atual.

Para criar propriedades de anotação, usam-se os editores de propriedade de anotação apropriados, disponíveis em cada uma das grandes seções (guias) do Protégé 5: **Active ontology**, **Classes**, **Object Properties**, **Data Type Properties**, **Individuals**. Para acessar a seus respectivos editores, é necessário clicar no ícone  **Annotation**, conforme a Figura 7.9:

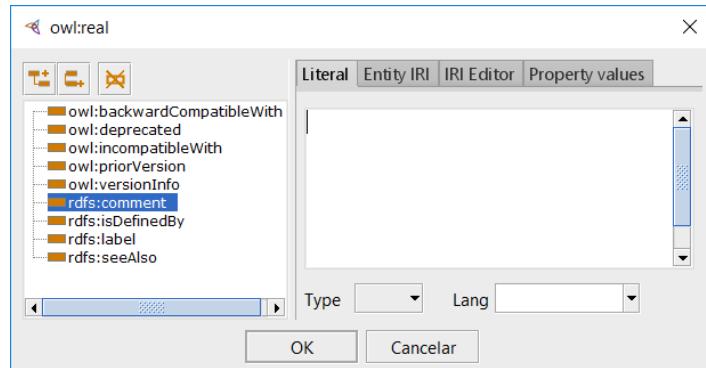


Figura 7.9 - editores de propriedade de anotação

É possível gerenciar as anotações usando a guia **Annotations properties** (Propriedades de anotações). Assim, novas propriedades de anotação podem ser criadas clicando no ícone , conforme apresentado na Figura 7.10:

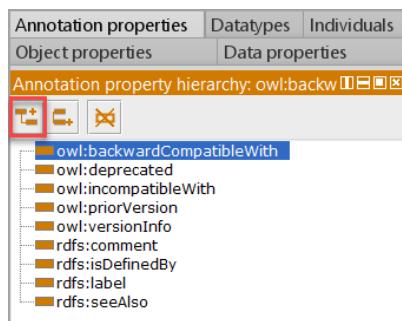


Figura 7.10 – Gerenciador das anotações usadas em OWL

7.5. Conjuntos múltiplos de Condições Necessárias e Suficientes

Em OWL, é possível ter vários conjuntos de condições necessárias e suficientes (*Equivalent classes*), como descrito na Figura 7.7:

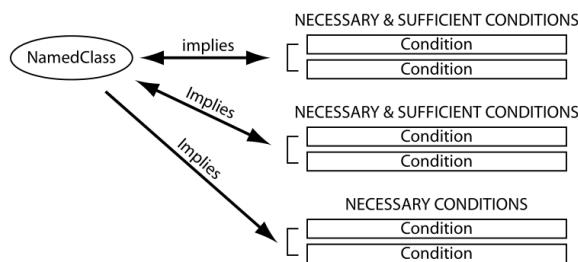


Figura 7.11 - Conjuntos múltiplos de condições necessárias e suficientes

Na guia **Class description** (Descrição de Classe), vários conjuntos de condições necessárias e suficientes são representados usando a seção **Equivalent To** (Equivalent class). Para criar um conjunto de condições necessárias e suficientes, usa-se o ícone  ao lado de **Equivalent To**. As classes equivalentes podem então ser digitadas na caixa de diálogo “Classe expression editor” que aparece.



No Protégé 5, as condições necessárias e suficientes são definidas na seção **Equivalente To** (Defined Classes) e as condições necessárias, na seção **SubClass Of** (Primitives Classes).

8. Apêndice A: Tipos de restrições (Restiction types)

Esse apêndice contém informações adicionais sobre os tipos de restrições para propriedades OWL. É indicado para leitores não familiarizados com noções de lógica, em que OWL se baseia.

Todos os tipos de restrições descrevem um conjunto sem nome que pode conter indivíduos. Esse conjunto corresponde a uma classe anônima. Quaisquer indivíduos membros dessa classe anônima satisfazem a restrição que descreve a classe (Figura 8.1). O termo "restrições" descreve restrições sobre relações através de propriedades, das quais participam indivíduos. Quando se descreve uma classe nomeada usando restrições, o que se faz realmente na prática é descrever uma superclasse anônima da classe nomeada.

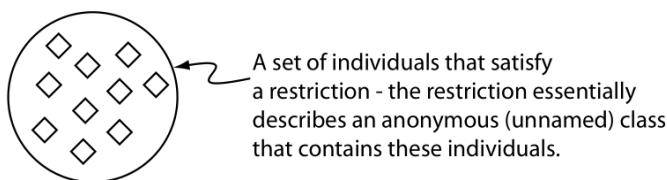


Figura 8.1 - Restrições que descrevem uma classe de indivíduos anônima

A Figura 8.1 esquematiza um conjunto de indivíduos que satisfazem uma restrição que descreve essencialmente uma classe anônima (sem nome) que contém esses indivíduos.

8.1. Restrições de quantificação (Quantifier Restrictions)

Restrições de quantificação possuem três partes:

- 1) O quantificador, que pode ser um quantificador existencial “some” (\exists) ou um universal “only” (\forall);
- 2) Uma propriedade, sobre a qual a restrição atua;
- 3) Um complemento da restrição (*filler*), que corresponde à classe nomeada cujos indivíduos atendem à restrição.

O quantificador impõe restrições sobre uma relação da qual um indivíduo participa. Isso ocorre porque o quantificador especifica que, **pelo menos, um** tipo de relação deve existir, ou especifica os tipos de relação que podem existir.

8.1.1. Restrições Existenciais (Existential Restrictions): `someValuesFrom`

As restrições existenciais, conhecidas como `someValuesFrom`, ou “**some**”, são representadas, na sintaxe da Lógica de descrição (DL syntax), pelo símbolo \exists (“some”). As restrições existenciais descrevem o conjunto de indivíduos que têm, **pelo menos, um** tipo específico de relação com indivíduos membros de uma classe.

A Figura 8.2 apresenta uma visão esquemática de uma restrição existencial `prop some ClassA` (ou $\exists \text{ prop ClassA}$)⁵⁰, ou seja, uma restrição sobre a propriedade `prop` com um descriptor `ClassA`. Observe que todos os indivíduos da classe anônima definida pela restrição têm pelo menos uma relação através da propriedade `prop` com um indivíduo que é membro da classe `ClassA`.

As linhas pontilhadas (Figura 8.2) representam o fato de que os indivíduos podem ter relações de tipo `prop` adicionais com outros indivíduos que não são membros da classe `ClassA`, mesmo que isso não tenha sido explicitamente estabelecido. A restrição existencial não restringe a relação `prop` a membros da classe `ClassA`, apenas estabelece que cada indivíduo deve ter, **pelo menos, uma relação `prop`** com um membro de `ClassA`. Essa característica recebe o nome de Open World Assumption - OWA (Pressuposição de mundo aberto).

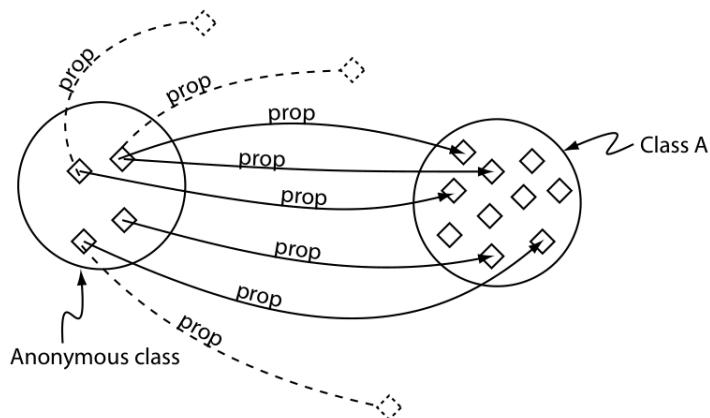


Figura 8.2 - Uma visão esquemática da restrição existencial (`prop some ClassA`)

Por exemplo, a restrição existencial `hasTopping some MozzarellaTopping` (ou $\exists \text{ hasTopping MozzarellaTopping}$) descreve o conjunto de indivíduos participantes de **pelo menos uma** ("some") relação `temRecheio` (`hasTopping`) com outro indivíduo, que é membro da classe `RecheioDeMuçarela` (`MozzarellaTopping`). Em linguagem natural, pode-se dizer que a restrição existencial é aquela que descreve as coisas que têm um recheio de muçarela.

O fato de a restrição existencial descrever um grupo de indivíduos que têm, pelo menos, um tipo de relação através da propriedade `temRecheio` (`hasTopping`) com um indivíduo membro da classe `RecheioDeMuçarela` (`MozzarellaTopping`) não significa que as relações apenas podem ocorrer através da propriedade `temRecheio` (`hasTopping`), ou apenas com indivíduo membro da classe `RecheioDeMuçarela` (`MozzarellaTopping`). Podem existir outras relações `temRecheio` (`hasTopping`) não especificadas de forma explícita.

⁵⁰ No Protégé 5 (e no Protégé 4), usa-se a sintaxe, por extenso, "some" (no meio da restrição): `prop some ClassA`. Nas versões anteriores de Protégé ou em outras aplicações, usa-se o símbolo lógico \exists (no início da restrição): $\exists \text{ prop ClassA}$

8.1.2. Restrições Universais (Universal Restrictions): allValuesFrom

As **restrições universais** são também conhecidas como **allValuesFrom**, ou “**only**”, uma vez que restringem o complemento da restrição (*filler*) de certa propriedade para uma classe específica. As restrições universais são representadas pelo símbolo \forall (“A” de cabeça para baixo). Descrevem o conjunto de indivíduos os quais, para uma dada propriedade, têm um tipo de relação apenas com outros indivíduos, membros de uma classe específica. Para uma dada propriedade, o conjunto de indivíduos descritos pela restrição universal vai também conter os indivíduos que não têm qualquer tipo de relação com essa propriedade, para qualquer outro indivíduo.

A restrição universal, apresentada na Figura 8.3, possui a propriedade **prop** e o complemento da restrição (*filler*) **ClassA**. Cabe destacar que uma restrição universal não garante a existência de um tipo de relação através da propriedade. Ela simplesmente estabelece que se existe um tipo de relação através da propriedade, então ela deve ocorrer com um indivíduo que é um membro da classe específica.

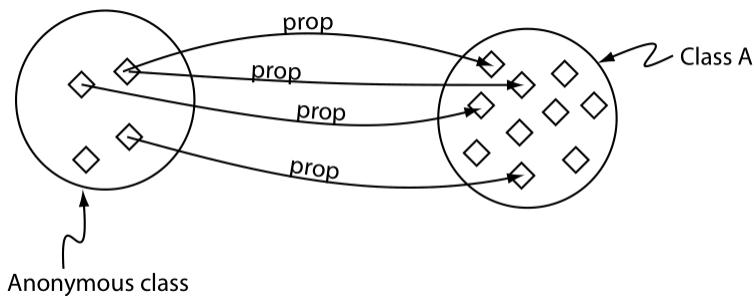


Figura 8.3 - Uma visão esquemática da restrição universal (*prop only ClassA*)

Por exemplo, a restrição `hasTopping only TomatoTopping` (ou $\forall \text{ hasTopping TomatoTopping}$) descreve a classe anônima de indivíduos que têm apenas (“*only*”) relações do tipo **temRecheio** (`hasTopping`) com indivíduos membros da classe **RecheioDeTomate** (`TomatoTopping`), ou, indivíduos que definitivamente não participam de nenhuma outra relação do tipo **temRecheio** (`hasTopping`).

8.1.3. Combinação de restrições Existenciais e Universais na descrição de classes

Um procedimento comum é combinar restrições universais e existenciais em definições de classes para certa propriedade.

Por exemplo, as seguintes restrições podem ser usadas em conjunto: `hasTopping some MozzarellaTopping` e `hasTopping only MozzarellaTopping` (ou $\exists \text{ hasTopping MozzarellaTopping} \wedge \forall \text{ hasTopping MozzarellaTopping}$).

Essa combinação descreve o conjunto de indivíduos que têm, pelo menos, uma relação do tipo **temRecheio** (`hasTopping`) com um indivíduo da classe **RecheioDeMuçarela** (`MozzarellaTopping`), e têm **apenas** (“*only*”) uma relação do tipo **temRecheio** (`hasTopping`) com indivíduos da classe **RecheioDeMuçarela** (`MozzarellaTopping`).

Não é comum, e em geral significa um erro, ao descrever uma classe, que se use a restrição universal junto à propriedade, sem uso da restrição universal correspondente junto à mesma propriedade.

No exemplo acima, caso se use apenas a restrição universal `hasTopping` **only** `MozzarellaTopping` (ou \forall `hasTopping` `MozzarellaTopping`), então se pode ter descrito o conjunto de indivíduos que apenas ("only") participa na relação `temRecheio` (`hasTopping`) com membros da classe `RecheioDeMuçarela` (`MozzarellaTopping`), e também aqueles indivíduos que não participam de qualquer relação de tipo `temRecheio` (`hasTopping`) (provavelmente, um erro).

8.2. Restrições `hasValue` (`hasValue` Restrictions)

Uma restrição `hasValue`, representada pelo símbolo \exists , descreve uma classe anônima de indivíduos que estão relacionados a outros indivíduos específicos por uma propriedade.

Trata-se de uma situação diferente daquela apresentada na restrição de quantificador, em que os indivíduos descritos pela restrição estão relacionados a qualquer indivíduo de uma classe específica através da propriedade específica. A Figura 8.4 mostra uma visão esquemática da restrição `hasValue` `prop` `value` `abc` (ou `prop` \exists `abc`)⁵¹.

Essa restrição descreve a classe anônima de indivíduos que têm, pelo menos, um tipo de relação através da propriedade `prop` com o indivíduo `abc`. As linhas pontilhadas (Figura 8.4) representam o fato de que, para um dado indivíduo, a restrição `hasValue` não restringe a propriedade usada na restrição de um tipo de relação com o indivíduo participante da restrição, isto é, podem existir outros tipos de relação com a propriedade `prop`.

Cabe notar que as restrições `hasValue` são semanticamente equivalentes a uma restrição existencial junto a uma propriedade `hasValue`, a qual tem um complemento da restrição (*filler*) que é uma classe enumerada que contém o indivíduo (e apenas ele) usado na restrição `hasValue`.

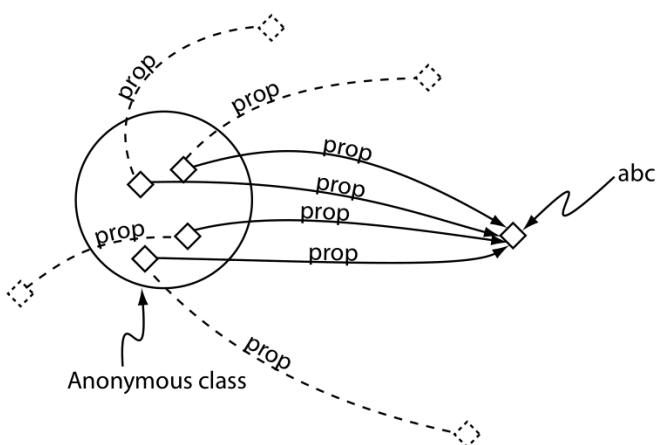


Figura 8.4 – Um exemplo de restrição `hasValue`

⁵¹ No Protégé 5 (e no Protégé 4), usa-se a sintaxe, por extenso, "value": `prop` `value` `abc`. Nas versões anteriores de Protégé ou em outras aplicações, usa-se o símbolo lógico \exists : `prop` \exists `abc`

8.3. Restrições de cardinalidade (*Cardinality Restrictions*)

As restrições de cardinalidade⁵² são usadas para explicitar o número de relações das quais um indivíduo pode participar em determinada propriedade. São conceitualmente simples e apresentadas em três formas: (1) restrições de cardinalidade mínima (*Minumum Cardinality Restrictions*); (2) restrições de cardinalidade máxima (*Maximum Cardinality Restrictions*); (3) restrições de cardinalidade exata (*Cardinality Restrictions*).

8.3.1. Restrições de cardinalidade mínima

As restrições de cardinalidade mínima especificam o número mínimo de relações das quais um indivíduo pode participar em determinada propriedade. O símbolo para a restrição de cardinalidade mínima é o maior ou igual a (\geq). Por exemplo, a cardinalidade mínima `hasTopping min 3 owl:Ting` (ou \geq `hasTopping 3`)⁵³ descreve os indivíduos (uma classe de anônimos que contém indivíduos) que participam em pelo menos três relações de tipo `hasTopping`.

As restrições de cardinalidade mínima não estabelecem limites máximos relativos ao número de relações que um indivíduo pode participar, em uma propriedade.

8.3.2. Restrições de cardinalidade máxima

As restrições de cardinalidade máxima especificam o número máximo de relações que um indivíduo pode participar em determinada propriedade. O símbolo para restrições de cardinalidade máxima é o menor ou igual a (\leq). Por exemplo, a cardinalidade máxima `hasTopping max 2 owl:Ting` (ou \leq `hasTopping 2`)⁵⁴ descreve a classe de indivíduos que participam no máximo de dois relações de tipo `hasTopping`.

Cabe notar que as restrições de cardinalidade máxima não estabelecem limites mínimos sobre o número de relações em que um indivíduo deve participar para uma determinada propriedade.

8.3.3. Restrições de cardinalidade exata

As restrições de cardinalidade especificam o número exato de relações em que um indivíduo deve participar para uma determinada propriedade. O símbolo para as restrições de cardinalidade é o igual (=). Por exemplo, a cardinalidade `hasTopping exactly 5` (ou $=$ `hasTopping 5`)⁵⁵ descreve o conjunto dos indivíduos (a classe anônima de indivíduos) que participam em exatamente 5 relações do tipo `hasTopping`.

⁵² Número de elementos de um conjunto. "cardinalidade", in Dicionário Priberam da Língua Portuguesa [em linha], 2008-2013, <https://www.priberam.pt/dlpo/cardinalidade> [consultado em 06-01-2018].

⁵³ No Protégé 5 (e no Protégé 4), usa-se a sintaxe, abreviada, "min" (no meio da restrição): `hasTopping min 3`. Nas versões anteriores de Protégé ou em outras aplicações, usa-se o símbolo lógico \geq (no início da restrição): \geq `hasTopping 3`

⁵⁴ No Protégé 5 (e no Protégé 4), usa-se a sintaxe, abreviada, "max" (no meio da restrição): `hasTopping max 3`. Nas versões anteriores de Protégé ou em outras aplicações, usa-se o símbolo lógico \leq (no início da restrição): \leq `hasTopping 3`

⁵⁵ No Protégé 5 (e no Protégé 4), usa-se a sintaxe, por extenso, "exactly" (no meio da restrição): `hasTopping exactly 5`. Nas versões anteriores de Protégé ou em outras aplicações, usa-se o símbolo lógico $=$ (no início da restrição): $=$ `hasTopping 5`

Cabe notar que a restrição de cardinalidade é, na verdade, uma forma simplificada de combinação entre as cardinalidades mínimas e máximas. Por exemplo, a restrição de cardinalidade acima pode ser representada como uma interseção de duas restrições (mínima e máxima): \leq hasTopping 5, and, \geq hasTopping 5.

8.3.4. Postulado de nome único (*Unique Name Assumption — UNA*) e cardinalidades

A linguagem OWL adota o Postulado de nome único (*Unique Name Assumption — UNA*), o que significa que diferentes nomes podem se referir ao mesmo indivíduo. Por exemplo, os nomes **Matt** e **Matthew** podem se referir ao mesmo indivíduo ou não.

As restrições de cardinalidade consideram indivíduos distintos e, dessa forma, é importante especificar que **Matt** e **Matthew** são o mesmo indivíduo, ou são diferentes.

Seja um indivíduo **Nick**, relacionado a indivíduos **Matt**, **Matthew** e **Matthew Horridge** por meio da propriedade **worksWith**. Também existe um indivíduo **Nick**, membro da classe de indivíduos que trabalham com no máximo dois outros indivíduos. Uma vez que OWL não adota o postulado de nome único (*Unique Name Assumption — UMA*), ao invés da situação resultar em um erro, a inferência automática indica que dois dos nomes se referem ao mesmo indivíduo. Ao afirmar que **Matt**, **Matthew** e **Matthew Horridge** são indivíduos diferentes, a base de conhecimento pode ser considerada inconsistente.

9. Apêndice B: Descrições de classes complexas (Complex Class Descriptions)

Uma classe OWL é especificada em termos de suas superclasses (subclasses)⁵⁶. Tais superclasses são em geral classes nomeadas e restrições, sendo que as restrições são de fato classes anônimas. As superclasses também podem tomar a forma de "descrições complexas", construídas a partir de descrições simples, conectadas por operadores lógicos. Os operadores mais conhecidos são:

- ◆ AND (\cap): a classe formada com o operador AND é conhecida como classe de interseção; a classe resultante é a interseção das classes individuais;
- ◆ OR (\cup): a classe formada com o operador OR é conhecida como classe de união; a classe resultante é a união das classes individuais.

9.1.1. Classes de interseção (\cap)

Uma classe de interseção é descrita pela combinação de duas ou mais classes, as quais utilizam o operador AND (\cap).

Por exemplo, seja a interseção de **Human** (humano) e **Male** (masculino), conforme Figura 9.1: descreve-se a classe anônima que contém indivíduos membros da classe **Human** e da classe **male**.

⁵⁶ No Protégé 5, usa-se "subclasse".

A semântica de uma classe interseção define que a classe anônima descrita é uma subclasse de **Human** e também uma subclasse de **Male**. Essa classe interseção anônima pode também ser usada em outra descrição de classes. Por exemplo, ao se construir uma descrição da classe **Man** especifica-se que **Man** é uma subclasse da classe anônima descrita pela interseção de **Human** e de **male**. Em outras palavras, **Man** é subclasse de **Human** e de **Man**.

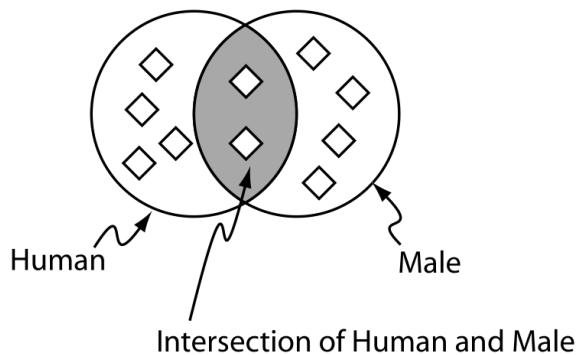


Figura 9.1 – Um exemplo de interseção ($\text{Human} \cap \text{Male}$) — A área escura representa a interseção

9.2. Classes de união (\cup)

A classe de união é criada pela combinação de duas ou mais classes usando o operador OR (\cup). Por exemplo, a união de **Man** e **Woman**, conforme a Figura 9.2. Ela descreve a classe anônima que contém os indivíduos pertencentes à classe **Man** ou à classe **Woman** (ou a ambas).

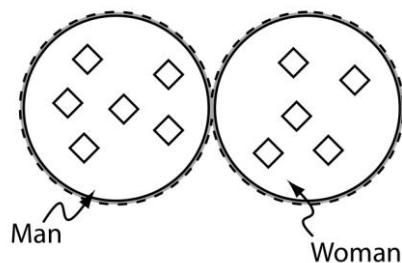


Figura 9.2 – Um exemplo de união ($\text{Man} \cup \text{Woman}$) — a área escura representa a união

A classe anônima descrita pode ser usada em outra descrição de classe. Por exemplo, a classe **Person** pode ser equivalente à união de **Man** e **Woman**.

10. Referências bibliográficas

HORRIDGE, M. et al. **A practical guide to building OWL ontologies using the Protégé-OWL plugin and CO-ODE tools (v.1.0)**. University Of Manchester, 2004.

HORRIDGE, M. **Um guia prático de construção de ontologias OWL**, Tradução de Almeida, M.B. e Soares, D.R. 2008.

HORRIDGE, M.; BRANDT, S. **A practical guide to building OWL ontologies using the Protégé-OWL plugin and CO-ODE tools (v.1.3)**. University of Manchester, 2011.