Pós-Graduação em Ciência da Computação

RENAN LEANDRO FERNANDES

Uma Ferramenta para Modelagem Visual de Ontologias com Raciocínio Automático

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

Recife
2019

RENAN LEANDRO FERNANDES

Uma Ferramenta para Modelagem Visual de Ontologias com Raciocínio Automático

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

Área de concentração: Inteligência Artificial

Orientador: Prof. Dr. Frederico Luíz Gonçalves de Freitas

Coorientador: Prof. Dr. Ryan Ribeiro de Azevedo

Recife
2019

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

F363f Fernandes, Renan Leandro
Uma ferramenta para modelagem visual de ontologias com raciocínio automático / Renan Leandro Fernandes. – 2019.
130 f.: il., fig., tab.

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

1. Inteligência artificial. 2. Web semântica. 3. Ontologias. I. Freitas, Frederico Luíz Gonçalves de (orientador). II. Título.

006.3 CDD (23. ed.) UFPE- MEI 2019-060

Renan Leandro Fernandes

“Uma Ferramenta para Modelagem Visual de Ontologias com Raciocínio Automático”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 14/02/2019.

_____

Orientador: Prof. Dr. Frederico Luiz Gonçalves de Freitas

BANCA EXAMINADORA

_____

Prof. Dr. Robson do Nascimento Fidalgo
Centro de Informática/UFPE
(**Examinador Interno**)

_____

Profa. Dra. Eunice Palmeira da Silva
Instituto Federal de Alagoas
(**Examinadora Externa**)

_____

Prof. Dr. Ryan Ribeiro de Azevedo
Departamento de Ciência da Computação/Campus Garanhuns
(**Co-Orientador**)

Dedico à minha mãe Josy por mostrar-me e incentivar-me ao caminho dos estudos e a meus avós por terem dado a ela a oportunidade de estudar. Estar aqui desenvolvendo este trabalho só foi possível graças ao esforço e carinho de vocês...

AGRADECIMENTOS

Meu agradecimento é sem tamanho para meus pais e meu irmão, que sempre estiveram ao meu lado e me deram suporte. Eu não teria conseguido chegar até aqui sem vocês.

Também sou muito grato à minha noiva Cássia Elaine por entender minhas ausências durante a realização deste trabalho e por me apoiar de uma forma que não tenho nem palavras para agradecer.

A todos os meus familiares e amigos por me incentivarem, e compreenderem minhas ausências durante minha pós-graduação.

Aos meus professores da graduação e da pós-graduação, sem exceção, todos foram essenciais para o meu crescimento intelectual. Deixo meu agradecimento especial aos professores Dr. Luciano Souza, Dr. Rodrigo Rocha, Dr. Ryan Azevedo e Dr. Fred Freitas. Aos dois últimos um obrigado especial pela partilha de conhecimento e o acolhimento no grupo de pesquisa. Serei eternamente grato.

Agradeço aos amigos que descobri durante o curso da graduação e continuamos levando essa amizade: Levy Souza, Israel Araújo e Diogo Espinhara. Obrigado pelas conversas relacionadas ao nosso curso e também as que não tinham tanta relação assim.

A todos os meus colegas de curso e de ônibus por todos nossos diálogos que de forma direta ou indireta me ajudaram a reforçar o que aprendi na instituição e com a vida.

Gostaria também de agradecer as demais pessoas que me ajudaram de forma indireta para que eu pudesse chegar a etapa final deste curso.

Por fim, mas não menos importante, agradeço ao CNPq por ter financiado os meus estudos através de uma bolsa. Agradeço também aos funcionários tanto do CIn-UFPE quanto da UFPE pelo suporte às minhas diversas necessidades.

RESUMO

A sintaxe e semântica formal das linguagens ontológicas da *Web* Semântica como *Web Ontology Language* (OWL) e *Resource Description Framework* (RDF) são de difícil aprendizado para pessoas sem conhecimento especializado na área de ontologias. As pesquisas em ontologias têm concentrado esforços na tentativa de minimizar este problema, buscando formas de ajudar desenvolvedores e engenheiros do conhecimento na construção de ontologias. Ferramentas de Autoria Inteligente, baseadas em Modelagem Visual, as quais possuem o intuito de facilitar as atividades dos usuários na construção dos seus artefatos computacionais, sejam eles *blogs*, *sites*, *softwares*, bases de dados, ontologias, entre outros, funcionam de forma simples e intuitiva. O intuito da dissertação é desenvolver uma ferramenta de autoria inteligente para construção, manipulação e manutenção de ontologias, permitindo sua criação a partir de modelagem visual. Outro objetivo desta dissertação foi utilizar o raciocínio automático de subsunção, deduzindo novos fatos a partir de fatos modelados e raciocínio de inconsistência durante a criação das ontologias, apresentando estes resultados de forma visual aos seus usuários. A ferramenta desenvolvida, denominada `Medina`, é útil no apoio das atividades de usuários experientes, como Engenheiros de Ontologias, no apoio a usuários leigos interessados no desenvolvimento de bases de conhecimento modeladas como ontologias, bem como em seu aprendizado. Em comparação com ferramentas de edição visual de ontologias encontradas na literatura, a ferramenta proposta foi a única a apresentar a *visual explanation* das deduções e inconsistências obtidas através do raciocínio automático. Na realização dos experimentos, foi percebido que utilizar a combinação de várias técnicas de visualização foi uma abordagem que apresentou bons resultados. A ferramenta `Medina` possui a visualização em grafos, em árvore e o uso de ocultar objetos visuais. Os usuários interagiram de maneira satisfatória entre as três técnicas com destaque para a funcionalidade de ocultar axiomas presente na ferramenta proposta e foi bem avaliada pelos participantes do experimento. Com a análise dos resultados, conclui-se que a ferramenta proposta possui capacidades de modelagem visual de ontologias com expressividade até $\mathcal{ALCHIQ}$, realiza raciocínios automáticos e também auxilia os desenvolvedores no processo de criação de ontologias, reduzindo a dificuldade empregada no desenvolvimento.

Palavras-chaves: Web Semântica. Modelagem Visual. Ontologias. Ferramenta de Autoria. Raciocínio Automático.

ABSTRACT

The syntax and semantics of the Semantic Web such as Web Ontology Language (OWL) and Resource Model Framework (RDF) are difficult to learn for people with no specialized knowledge in ontology area. Research on ontologies has focused efforts to minimize this problem, seeking ways to help developers and knowledge engineers in the construction of ontologies. Intelligent Authoring Tools, based on Visual Modeling, which are intended to facilitate the activities of users in the construction of their computational artifacts, be they blogs, sites, software, databases data, ontologies, among others, work in a simple and intuitive way. The purpose of the dissertation is to develop a tool for intelligent authoring for the construction, manipulation, and maintenance of ontologies, allowing it's creation from visual modeling. Another objective of this dissertation was to use automatic subsumption reasoning, deducing new facts from modeled facts and inconsistency reasoning during the creation of ontologies, presenting these results in a visual way to its users. The developed tool, named Medina, is useful in supporting the activities of experienced users, such as Ontology Engineers, in supporting lay users interested in developing knowledge bases modeled as ontologies, as well as in their learning. In comparison with the visual editing tools of ontologies found in the literature, the proposed tool was the only one to present a visual explanation of the deductions and inconsistencies obtained through automatic reasoning. In the accomplishment of the experiments, it was perceived that to use the combination of several visualization techniques was an approach that presented good results. The Medina tool has graph view, tree view, and the use of hiding visual objects. The users interacted satisfactorily among the three techniques, highlighting the functionality of hiding axioms present in the proposed tool and was well evaluated by the participants of the experiment. The analysis of the results shows that the proposed tool has visual modeling capabilities of ontologies with expressivity up to $\mathcal{ALCHIQ}$, performs automatic reasoning and also assists developers in the process of creating ontologies, reducing the difficulty employed in ontology development.

Keywords: Semantic Web. Visual Modeling. Ontologies. Authorship tool. Automatic Reasoning.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE SIGLAS

API	*Application Programming Interface*
DL	*Description Logics*
GUI	*Graphical User Interface*
IA	Inteligência Artificial
JSON	*JavaScript Object Notation*
JUNG	*Java Universal Network Graph Framework*
OWL	*Web Ontology Language*
RDF	*Resource Description Framework*
UML	*Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTO E MOTIVAÇÃO

Os seres humanos, mesmo que de maneira inconsciente, geralmente tomam decisões baseadas em seu conhecimento sobre o mundo. A inteligência humana é alcançada através de um conhecimento interno — seja sobre verdades ou crenças a respeito do mundo — utilizando-o para obter novas conclusões através de um processo de raciocínio (RUSSELL; NORVIG, 2009, Capítulo 7).

Na Inteligência Artificial (IA), sistemas são desenvolvidos utilizando processos de raciocínio computacionais, obtendo novos fatos a partir de conhecimentos prévios armazenados em sua base de conhecimento. Esses sistemas baseados em conhecimento (chamados desta forma por utilizarem uma base de conhecimento) necessitam que o conhecimento utilizado seja representado de uma maneira adequada às decisões que devem ser tomadas. Segundo Brachman e Levesque (2004), Representação do Conhecimento é a área da IA que se preocupa com a forma como o conhecimento pode ser representado simbolicamente e manipulado de forma automatizada por programas de raciocínio.

A representação de conhecimento pode ser implementada de diferentes formas, dentre elas prevalecem as redes semânticas, regras e a lógica (GRIMM; HITZLER; ABECKER, 2007). Uma rede semântica é um grafo em que os nós representam conceitos e os arcos representam as relações entre esses conceitos. Regras vêm na forma de construções SE-ENTÃO e permitem expressar vários tipos de declarações complexas e podem ser encontradas em sistemas de programação lógica, bancos de dados dedutivos ou sistemas de regras de negócios. Tanto as redes semânticas como as regras foram formalizadas usando a lógica para dar-lhes uma semântica precisa. Sem uma formalização tão precisa, elas são ambíguas e consequentemente problemáticas para fins computacionais (GRIMM; HITZLER; ABECKER, 2007).

Ao desenvolver um sistema baseado em conhecimento, os engenheiros do conhecimento utilizam fatos relacionados a um domínio específico. Como forma de codificar o conhecimento sobre um certo domínio, e que seja processável por uma máquina, são utilizadas ontologias.

Ontologias são especificações formais explícitas de conceitualizações compartilhadas de um domínio de interesse (GRUBER, 1995). Ela é formal, pois é expressa em uma linguagem de representação de conhecimento; explícita, pois o conhecimento está acessível por máquinas; possui conceitualizações, uma vez que seu desenvolvimento é através de símbolos que representam conceitos e suas relações; e compartilhada por refletir uma concordância entre pessoas a respeito do domínio especificado. Na seção a seguir são apresentados os problemas da pesquisa.

1.2 PROBLEMA

As ontologias até agora não conseguiram obter ampla adesão e um dos principais motivos é a sua curva de aprendizado. O desenvolvimento de ontologias não é uma tarefa trivial e muitas vezes é um processo custoso em termos de tempo e finança (DENAUX, 2013).

A criação de ontologias é uma tarefa desafiadora que requer especialização em engenharia de conhecimento, experiência em lógica e conhecimento do domínio (DENAUX et al., 2012). No processo de desenvolvimento de ontologias, os engenheiros de conhecimento são os responsáveis por desenvolver as ontologias. Porém nem sempre são esses engenheiros que possuem o conhecimento necessário para modelar a ontologia. Para o desenvolvimento da ontologia é necessário então um ou mais especialistas com conhecimento do domínio a ser modelado.

A abordagem conduzida pelo engenheiro do conhecimento pode dificultar o processo de construção da ontologia, pois o conhecimento do domínio pode se tornar secundário ao processo de modelagem eficiente do conhecimento (DIMITROVA et al., 2008). Isto é especialmente verdade onde o especialista do domínio não tem conhecimento das linguagens e ferramentas usadas para construir a ontologia. Há, portanto, uma necessidade crescente de ferramentas de autoria de ontologias que forneçam interfaces intuitivas para inserir construções de ontologias (DENAUX et al., 2012) de maneira que seja atrativa e útil também para o especialista do domínio.

A experiência em lógica necessária na criação de ontologias se deve a sua formalização. Um dos formalismos mais utilizados para a representação de ontologias é a Lógica de Descrições (NARDI; BRACHMAN, 2003) ou DL (do inglês *Description Logics*). A DL é uma fração decidível da lógica de primeira ordem, dotada de uma semântica formal, permitindo que motores de raciocínio possam ser desenvolvidos e aplicados em ontologias com esta representação de conhecimento. A inferência básica em expressões de conceitos em DL é a subsunção, escrita em notação de fórmula como $C \sqsubseteq D$. Determinar uma subsunção é verificar que o conceito D é mais geral que o conceito C (NARDI; BRACHMAN, 2003).

Ontologias em DL são representadas em duas estruturas: TBox e ABox (BAADER; NUTT, 2003). Na TBox estão armazenados os fatos, também chamados de axiomas, referentes a terminologias do domínio, como por exemplo definições e relações entre conceitos. Na ABox estão contidos fatos relacionados a instâncias da ontologia, também chamadas de indivíduos. No processo de raciocínio em ontologias representadas em DL, uma ontologia é dita inconsistente quando os fatos presentes em sua TBox e ABox estão em contradição.

O processo de raciocínio em ontologias é uma funcionalidade relevante em ontologias de domínio. Entender quais fatos inferem novos conhecimentos ajuda os engenheiros de conhecimento a validar se os fatos adicionados estão ou não corretos.

Ferramentas, métodos, princípios e processos de construção de ontologias, principal camada da Web Semântica (BERNERS-LEE; HENDLER; LASSILA, 2001), têm sido propostos

e pesquisados nas últimas décadas, como forma de capturar, representar e compartilhar o conhecimento das pessoas a respeito de domínios de conhecimento de maneira que possam ser entendidos e processados por sistemas de informação.

A ferramenta mais utilizada atualmente por engenheiros de ontologia, o `Protégé` (MUSEN, 2015), é uma ferramenta que fornece visões gerais de porções da ontologia em algum formato gráfico. No entanto, ele tem poucos ou nenhum recurso de edição gráfica, exigindo que os projetistas sejam capazes de especificar ontologias em termos de fórmulas lógicas, especialmente para modelar axiomas complexos (LEMBO et al., 2016).

Nos últimos anos, diversos modelos de representação de conhecimento foram concebidos. Nenhum deles, conseguiu se estabelecer como uma linguagem visual de referência (LEMBO et al., 2016). Sistemas que utilizam de linguagens visuais para criação e manutenção de ontologias também foram concebidos, tais como `Eddy` (LEMBO et al., 2016), e `OWLGrEd` (BĀRZDIŅŠ et al., 2010). `Eddy` é, segundo Lembo et al. (2016), o primeiro editor de que permite criar ontologias usando apenas funcionalidades gráficas. O `OWLGrEd` utiliza de uma linguagem visual estendida da UML (*Unified Modeling Language*), permitindo a visualização de fragmentos da ontologia (LIEPINŠ; CERANS; SPROGIS, 2012) e uma representação textual de conceitos.

Além de representar e editar ontologias visualmente, uma ferramenta deve permitir que o usuário possa entender as consequências lógicas dos fatos da ontologia que está sendo desenvolvida. Nesse sentido, existem algumas ferramentas desenvolvidas, como `OntoTrack` (LIEBIG; NOPPENS, 2005) e `AvOnEd` (HALLAY et al., 2017), capazes de apresentar ao usuário inferências ou não permitir a inserção de fatos inconsistentes.

O `OntoTrack` permite verificar quais axiomas implicam em raciocínios através de *explanations*[1] textuais (LIEBIG; NOPPENS, 2005). Uma *explanation* é simplesmente o conjunto preciso de axiomas em uma ontologia responsável por uma implicação específica (KALYANPUR et al., 2007). No entanto, `OntoTrack` e `AVOnEd` não apresentam *explanations* visuais, ou *visual explanations*, das inferências obtidas, funcionalidade útil para que o desenvolvedor da ontologia entenda quais conjuntos de axiomas causam implicações lógicas de maneira visual.

Capturar fatos advindos de usuários, representá-los em formalismo lógico e realizar raciocínio (para este trabalho, é usado o raciocínio dedutivo, sobre um subconjunto decidível da lógica de predicados) automático sem ou com pouca intervenção humana são desafios e, consequentemente apontam à solução descrita nesta dissertação, combinando as seguintes áreas de pesquisa: Ontologias (FREITAS; STUCKENSCHMIDT; NOY, 2005), Representação do Conhecimento e Raciocínio Automático (BAADER et al., 2003).

Na Figura 1 é apresentado um exemplo de representação visual de conhecimento na ferramenta proposta nesta dissertação, chamada `Medina`, onde estão modelados axiomas referentes ao domínio de pizzas. É possível visualizar os conceitos e suas relações além

[1] Para evitar problemas de interpretação com a tradução do termo, foi decidido utilizá-lo em inglês

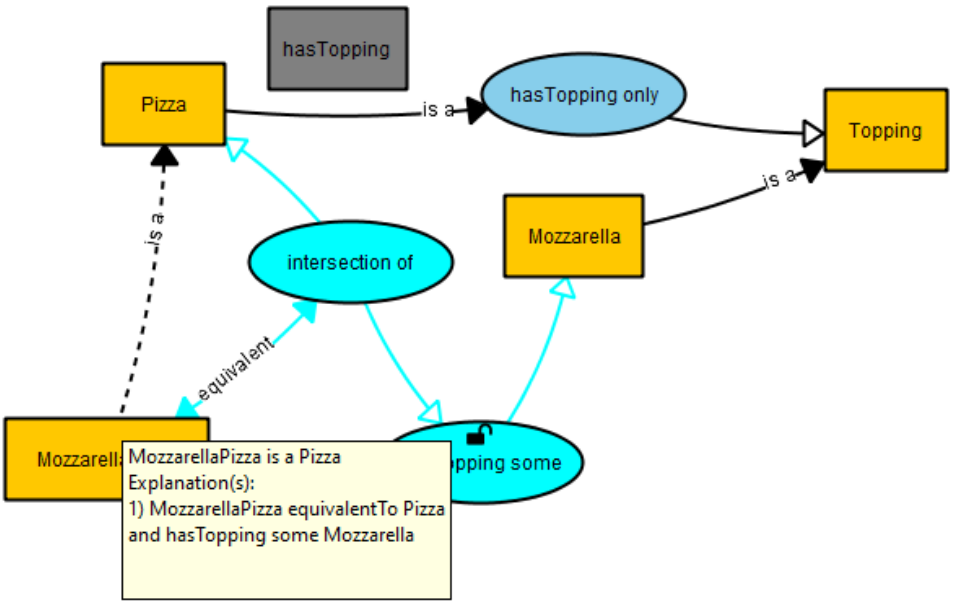


Figura 1 – Representação visual de um fragmento de ontologia com domínio sobre pizzas. Na representação foi inferido, através de raciocínio automático, que $MozzarellaPizza \sqsubseteq Pizza$, sendo destacados os fatos contidos na ontologia que causam essa inferência.

de verificar em destaque (na cor azul ciano) os fatos contidos na ontologia que causam essa inferência (a *visual explanation* da inferência). O mesmo fragmento de ontologia da Figura 1 pode ser visto representado em fórmulas lógicas de DL:

$$MozzarellaPizza \equiv \exists hasTopping.Mozzarella \sqcap Pizza \tag{1.1}$$

$$Mozzarella \sqsubseteq Topping \tag{1.2}$$

No `Medina` a construção dos fatos de uma ontologia acontece de maneira visual, através da ligação de arcos entre os componentes visuais, diferentemente da ferramenta `Protégé` que utiliza uma notação textual para representação dos fatos. Na adição de novos fatos, é executado um raciocínio automático, detectando novos fatos ou contradições na ontologia desenvolvida, apresentando as inferências e suas *visual explanations*.

Diante do contexto e problemas mencionados, a ferramenta desenvolvida proporciona aos seus usuários, construir através de modelagem visual, ontologias expressivas e livres de fatos contraditórios. Assim é possível construir e obter como resultado axiomas com os principais construtores que a Lógica de Descrições (DL) admite, como (BAADER, 2003): conjunção ($\sqcap$), disjunção ($\sqcup$), negação ($\neg$), restrição existencial ($\exists$) e restrição universal ($\forall$). A seguir apresentamos nossos objetivos.

1.3 OBJETIVOS

O objetivo geral deste trabalho é demonstrar que uma ferramenta de autoria inteligente para construção, manipulação e modificação de ontologias, que permite sua criação a partir de modelagem visual e realize raciocínio automático, apresentando também visualmente as *explanations* desses raciocínios é uma solução viável para o desenvolvimento de ontologias.

A este objetivo geral correspondem os objetivos específicos indicados a seguir:

- Demonstrar que a ferramenta de autoria inteligente denominada `Medina` é capaz de:
  - Construir ontologias em OWL DL com expressividade de alto nível (máxima $\mathcal{ALCHIQ}$) a partir de modelagem visual;
  - Realizar raciocínio de subsunção deduzindo que classes são subclasses de outras, a partir de suas respectivas descrições;
  - Detectar e verificar inconsistências em tempo de desenvolvimento nas ontologias construídas;
  - Apresentar *visual explanations* das deduções obtidas em tempo de desenvolvimento;
  - Mitigar dificuldades inerentes a construção manual e não visual de ontologias;
- Permitir que usuários acelerem o desenvolvimento das bases de conhecimento modeladas como ontologias, no qual são gerados automaticamente códigos OWL DL a partir dos modelos visuais.

1.4 RELEVÂNCIA E IMPACTO DO PROJETO

Os impactos e a relevância do projeto para o desenvolvimento científico e tecnológico foram:

a) A construção efetiva de uma ferramenta de autoria inteligente, com capacidades de modelagem visual e raciocínio automático de forma a permitir a reutilização dos recursos — componentes e demais artefatos desenvolvidos — entre os vários membros da comunidade de pesquisa; e

b) O uso de uma ferramenta de autoria inteligente com capacidades de visualização e edição de ontologias visualmente possibilita a constituição de um ambiente interativo de desenvolvimento de ontologias, permitindo o intercâmbio de práticas entre os agentes participantes do desenvolvimento e a investigação científica e computacional.

1.5 ESTRUTURA DO TRABALHO

O trabalho está dividido em seis capítulos. Apresentamos no Capítulo 2 a Fundamentação Teórica, abordando conceitos como Web Semântica, Ontologias, Motor de Inferência e Modelagem Visual. No Capítulo 3 está descrita a ferramenta `Medina`, apresentando sua arquitetura, componentes e funcionalidades desenvolvidas. No Capítulo 4 apresentamos os experimentos realizados, as avaliações quantitativas e qualitativas e os resultados obtidos. No Capítulo 5 apresentamos os trabalhos relacionados, ferramentas que se assemelham a ferramenta proposta. Nesse capítulo também é feito um comparativo entre essas ferramentas. Por fim, no Capítulo 6 apresentamos as conclusões e os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados detalhes de alguns conceitos presentes nesta dissertação. São descritas as tecnologias que fazem parte da *Web* Semântica, tais como ontologias, as linguagens OWL e RDF *Schema*. Além disso, são descritos detalhes sobre a sintaxe, semântica e famílias da Lógica de Descrições, formalismo lógico utilizado por diversos engenheiros de ontologia e no qual a linguagem OWL foi inspirada. Neste capítulo também são descritas as diversas formas de representação visual de ontologia, apresentando técnicas utilizadas por outros sistemas, suas vantagens e desvantagens.

2.1 WEB SEMÂNTICA

Ao longo dos anos, a *Web* passou a processar uma quantidade de dados cada vez maior. Grande parte do processamento de dados é realizado por programas que não só buscam informações em várias páginas como também interagem na *Web*, gerando informações.

Para que os programas que processam dados na Web possam realizar tarefas de acordo não só com as estruturas das páginas e dos textos nelas inseridos, mas que também fosse possível que os programas entendessem o significado dos documentos, foi desenvolvida a *Web* Semântica (BERNERS-LEE; HENDLER; LASSILA, 2001), como uma extensão da *Web* até então.

Para tornar a *Web* mais acessível para os computadores, deve-se levar em consideração princípios como: disponibilizar dados em formatos padronizados; disponibilizar relações sobre os indivíduos e não somente o conjunto de dados; e descrever a semântica dos dados em algum formalismo (ANTONIOU et al., 2012, Capítulo 1).

Para uma prática real desses princípios a W3C[1] (*World Wide Web Consortium*), consórcio responsável pelo desenvolvimento de padrões para a *Web*, teve um papel crucial. Em relação a estruturação e padronização dos dados foi estabelecido o uso de XML[2] como um padrão. As relações entre os indivíduos e a semântica dos dados podem ser obtidas com outras duas linguagens padronizadas pela W3C: RDF *Schema* e OWL, que serão descritas nas próximas seções. A seguir é descrita a Lógica de Descrições.

2.2 LÓGICA DE DESCRIÇÕES

As Lógicas de Descrição ou DLs são formalismos para a representação de conhecimento, derivados da lógica de primeira ordem (CAI; MING; LI, 2008). As DLs podem ser usadas para representar o conhecimento de domínios de aplicações, de maneira estruturada e formal (BAADER; HORROCKS; SATTLER, 2005).

[1] https://www.w3.org/
[2] https://www.w3.org/XML/

É apresentado nas subseções seguintes os componentes, construtores, semântica e as tarefas de raciocínio da Lógica de Descrições, especificamente da DL $\mathcal{ALCHIQ}$.

2.2.1 Componentes

A representação de conhecimento em DL é feita em duas estruturas: TBox e ABox.

A TBox está associada a conhecimento terminológico, relacionado a conjuntos de indivíduos e após o desenvolvimento da base de conhecimento, raramente são alterados (NARDI; BRACHMAN, 2003). Na TBox o conhecimento é inserido através de axiomas terminológicos ($\sqsubseteq$) ou de definição ($\equiv$) (BAADER; NUTT, 2003). Axiomas são proposições consideradas verdadeiras para o domínio que está sendo modelado.

Na ABox são inseridos os axiomas relacionados às instâncias do conhecimento a ser representado, tratando-se de uma estrutura cujo conhecimento representado é específico e que geralmente são alterados (NARDI; BRACHMAN, 2003). Os axiomas inseridos na ABox são axiomas de asserção, indicando instâncias de conceitos ($C(a)$) ou de relações ($R(a, b)$) (BAADER; NUTT, 2003). Considere os seguintes axiomas:

$$Gato \sqsubseteq Mamifero \tag{2.1}$$

$$Gato(darwin) \tag{2.2}$$

O Axioma 2.1 é um axioma terminológico cujo significado representa que todos os indivíduos que são gatos, também são mamíferos. Mas o contrário não necessariamente é verdade. Já o Axioma 2.2 é um axioma de asserção que indica que o indivíduo *darwin* é um gato. De acordo com o vocabulário da Lógica de Descrições, Gato e Mamifero são **conceitos** e *darwin* é um **indivíduo**.

A seguir são apresentadas as linguagens de descrição que compõem a Lógica de Descrições.

2.2.1.1 Descrições das linguagens

As linguagens de descrição são fragmentos da Lógica de Descrições, separadas de acordo com os construtores disponíveis na linguagem. A linguagem mais básica é a linguagem $\mathcal{AL}$ (BAADER; NUTT, 2003). Sejam C e D conceitos, os conceitos podem ser descritos em $\mathcal{AL}$ da seguinte forma:

$$
\begin{aligned}
C, D \longrightarrow A| & \ (conceito\ atômico)\\
\top| & \ (conceito\ superior)\\
\bot| & \ (conceito\ inferior)\\
\neg A| & \ (negação\ atômica)\\
C \sqcap D| & \ (interseção)\\
\forall R.C| & \ (restrição\ universal)\\
\exists R.\top| & \ (restrição\ existencial\ limitada)
\end{aligned}
$$

Perceba que em $\mathcal{AL}$ só é permitida a negação em conceitos atômicos e o escopo da restrição existencial é somente para o conceito superior (BAADER; NUTT, 2003). Os conceitos superior e inferior estão relacionados a hierarquia de conceitos utilizada na Lógica de Descrições. $\top$ representa um conceito no qual todos os outros conceitos estão abaixo na hierarquia, enquanto $\bot$ representa um conceito no qual todos os outros conceitos estão acima dele na hierarquia de conceitos. Uma das tarefas de raciocínio possíveis em DL é justamente determinar em que parte da hierarquia está um determinado conceito. Supondo que *Gato*, $Macho$ e $Mamifero$ são conceitos atômicos e que $temFilhos$ é uma propriedade atômica, os seguintes conceitos complexos em DL pertencem a linguagem $\mathcal{AL}$:

$$Gato \sqcap Macho \tag{2.3}$$

$$\forall temFilho.Gato \tag{2.4}$$

$$\exists temFilho.\top \tag{2.5}$$

$$(Gato \sqcap Macho) \sqcap \neg Mamifero \tag{2.6}$$

É importante notar que a utilização de parênteses é permitida na construção de conceitos complexos, conforme pode ser visto no Axioma 2.6.

É possível ainda estender a linguagem $\mathcal{AL}$, acrescentando novas possibilidades de construção de conceitos complexos. Acrescentando o uso de negação não só para conceitos atômicos, mas também para conceitos complexos é acrescentado $\mathcal{C}$ (complemento) à linguagem $\mathcal{AL}$ definindo assim $\mathcal{ALC}$. Os construtores de união e restrição existencial completa estão também definidos em $\mathcal{ALC}$ uma vez que é possível assumir que se uma linguagem possui negação de conceitos complexos, interseção e restrição universal completa, é possível obter união e a restrição existencial completa a partir deles (BAADER; NUTT, 2003).

Os quantificadores existencial e universal não possibilitam expressar o número máximo, mínimo ou igual de vezes que uma relação pode ser realizada na definição de um conceito

complexo. Adicionando essa nova possibilidade a linguagem é possível também obter a linguagem $\mathcal{ALCQ}$, sendo $\mathcal{Q}$ representante das restrições de número qualificadas[3].

Até o momento as propriedades utilizadas em $\mathcal{ALCQ}$ são propriedades atômicas. É possível também estender a possibilidade de construtores das propriedades e alcançar uma expressividade maior. Utilizar uma hierarquia de propriedades, acrescenta símbolos e se obtém a linguagem $\mathcal{ALCHQ}$, e permitir o uso de propriedades inversas acarreta na linguagem $\mathcal{ALCHIQ}$. Considere as propriedades atômicas $temFilho$, $temGato$ e $ehFilhoDe$, uma base de conhecimento com expressividade $\mathcal{ALCHIQ}$ suporta os seguintes axiomas:

$$temGato \sqsubseteq temFilho \tag{2.7}$$

$$temFilho \equiv ehFilhoDe^{-} \tag{2.8}$$

O Axioma 2.7 representa que todas as relações $temGato$ são também relações $temFilho$, ou seja, todo indivíduo que teve um gato teve também um filho[4]. Com a construção de relações com hierarquia é possibilitado aos desenvolvedores construir bases de conhecimento cada vez mais ricas e expressivas. Já no Axioma 2.8 é afirmado que toda relação $temGato$ possui como **relação inversa** $ehfilho$.

Até o momento foram apresentados detalhes da DL, suas famílias de linguagens, a terminologia utilizada para representá-las e os construtores possíveis de se utilizar em cada linguagem. Entretanto, é importante também entender o significado de cada um dos construtores e dos axiomas. Detalhes sobre a semântica são apresentados na próxima subseção.

2.2.2 Semântica

A Lógica de Descrições foi desenvolvida com base nas redes semânticas e na lógica de primeira ordem (NARDI; BRACHMAN, 2003). Por se tratar de uma linguagem de representação de conhecimento lógica, possui a necessidade de uma semântica bem definida e não ambígua.

Na subseção anterior foram vistas as famílias de linguagens pertencentes a Lógica de Descrições. A principal motivação para a separação das linguagens dessa maneira é a complexidade de raciocínio para cada um daqueles construtores. Por exemplo: implementar uma tarefa de raciocínio para uma base de conhecimento em $\mathcal{AL}$ é mais simples que para uma base representada em $\mathcal{ALC}$, por isso $\mathcal{AL}$ suporta apenas a negação atômica.

Para definir a semântica de conceitos da linguagem $\mathcal{ALCHIQ}$, considere interpretações $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ que consistem em um conjunto não vazio $\Delta^{\mathcal{I}}$, onde $\Delta^{\mathcal{I}}$ é o domínio da interpretação e uma função de interpretação que atribui para cada conceito atômico A um

[3] Por trabalharmos tanto com $\mathcal{ALCN}$ quanto $\mathcal{ALCQ}$, foi escolhido apresentar a família com maior expressividade.

[4] Considerando o termo filho como descendente de outro indivíduo.

conjunto $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ e para cada propriedade atômica R uma relação binária $R \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. A função de interpretação pode ser estendida para cada um dos conceitos complexos (BAADER; NUTT, 2003) vistos em 2.2.1.1 (BAADER, 2003):

$$\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} | \forall b.(a,b) \in R^{\mathcal{I}} \longrightarrow b \in C^{\mathcal{I}}\} \\
(\exists R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} | \exists b.(a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \\
(\geq n\ R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} | \#\{b \in \Delta^{\mathcal{I}} | (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \geq n\} \\
(\leq n\ R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} | \#\{b \in \Delta^{\mathcal{I}} | (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \leq n\} \\
(= n\ R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} | \#\{b \in \Delta^{\mathcal{I}} | (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} = n\}
\end{aligned}$$

Com a estensão da função de interpretação para os conceitos complexos vistos foi possível obter a semântica da linguagem $\mathcal{ALCQ}$. É possível também estender a função de interpretação para as propriedades afim de obter também a semântica das propriedades inversas:

$$(R^{-})^{\mathcal{I}} = \{(b,a) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} | (a,b) \in R^{\mathcal{I}}\} \tag{2.9}$$

Como visto em 2.2.1.1, uma base de conhecimento representada em DL possui duas estruturas: TBox e ABox. A sintaxe dessas estruturas foi vista em 2.2.1.1, sendo necessária agora sua representação semântica. Considerando ainda a função de interpretação definida anteriormente e sejam C e D conceitos, R e S propriedades e a um indivíduo, temos que (BAADER, 2003):

$$\begin{aligned}
(C \sqsubseteq D)^{\mathcal{I}} &= C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \\
(C \equiv D)^{\mathcal{I}} &= C^{\mathcal{I}} = D^{\mathcal{I}} \\
(R \sqsubseteq S)^{\mathcal{I}} &= R^{\mathcal{I}} \subseteq S^{\mathcal{I}}
\end{aligned}$$

Além de levar em consideração a semântica formal das linguagens da Lógica de Descrições é importante, antes de descrever as tarefas de raciocínio, levar em consideração a semântica de mundo aberto da ABox. Normalmente os sistemas de representação de conhecimento são aplicados em situações em que não se pode supor que o conhecimento na

base de conhecimento esteja completo (semântica do mundo fechado) (BAADER; NUTT, 2003), mesmo que a ABox se assemelhe a um banco de dados relacional, formado por relações binárias e unárias.

Pode-se então dar uma semântica para a ABox estendendo a interpretação para nomes de indivíduos (BAADER; NUTT, 2003). Portanto, uma interpretação $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ não mapeia apenas conceitos e propriedades atômicas para conjuntos e relações, mas também mapeia cada nome de indivíduo a para um elemento $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. Nesse caso, podemos definir a semântica para os axiomas da ABox da seguinte forma:

$$(C(a))^{\mathcal{I}} = a^{\mathcal{I}} \in C^{\mathcal{I}}$$
$$(R(a,b))^{\mathcal{I}} = (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$$

Uma interpretação satisfaz a ABox $\mathcal{A}$ se ela satisfaz cada asserção em $\mathcal{A}$. Quando isso ocorre, é dito que a interpretação $\mathcal{I}$ é um **modelo** de asserções da ABox $\mathcal{A}$ (BAADER; NUTT, 2003).

2.2.3 Tarefas de Raciocínio

Um sistema de representação de conhecimento baseado em DLs é capaz de executar tipos específicos de raciocínio. O propósito de um sistema de representação de conhecimento vai além de armazenar definições e asserções de conceitos, uma base de conhecimento tem uma semântica que a torna equivalente a um conjunto de axiomas na lógica de predicados de primeira ordem (BAADER; NUTT, 2003). Assim, como qualquer outro conjunto de axiomas, ele contém conhecimento implícito que pode ser explicitado através de inferências.

Ao modelar um domínio é construída uma terminologia a partir da definição de conceitos. É importante, durante esse processo, descobrir se um novo conceito adicionado é contraditório (BAADER; NUTT, 2003).

Uma das inferências mais importantes a ser obtida é a checagem de satisfatibilidade. Considere uma terminologia $\mathcal{T}$, se existe um modelo de $\mathcal{T}$ em que um conceito é um conjunto não vazio naquela interpretação, então esse conceito é *satisfatível* em relação a $\mathcal{T}$. Caso contrário, o conceito é dito *insatisfatível*. Considere os seguintes axiomas:

$$C \equiv D \sqcap E \tag{2.10}$$
$$C \sqsubseteq \neg D \tag{2.11}$$

Uma terminologia T que possua o Axioma 2.10 e o Axioma 2.11 não admite interpretações em que o conceito C seja satisfatível, devido a semântica dos construtores $\sqcap$ e $\neg$ dado que os indivíduos que fazem parte do conceito C fazem parte também dos conceitos D e E e não podem ser conceitos de D, de acordo com o Axioma 2.11.

A partir da checagem de satisfatibilidade é possível obter outros tipos de inferência: subsunção, equivalência e disjunção. Sejam os conceitos C e D, o conceito C é **subsumido** pelo conceito D em relação a $\mathcal{T}$ se $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ para todos os modelos $\mathcal{I}$ de $\mathcal{T}$ (BAADER; NUTT, 2003). O conceito C é subsumido por D e E, de acordo com o Axioma 2.10.

Dois conceitos são ditos **equivalentes** em relação a $\mathcal{T}$ se $C^{\mathcal{I}} = D^{\mathcal{I}}$ para todos os modelos $\mathcal{I}$ de $\mathcal{T}$ (BAADER; NUTT, 2003). Considerando uma base de conhecimento com o axioma $D \sqsubseteq C$ e o Axioma 2.10, poderia-se concluir que C e D são equivalentes, já que é possível obter por raciocínio que C é subsumido por D.

A **disjunção** entre dois conceitos em relação a $\mathcal{T}$ ocorre se $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ para todos os modelos $\mathcal{I}$ de $\mathcal{T}$ (BAADER; NUTT, 2003). Em casos onde a TBox relacionada as inferências está clara, pode-se omitir a qualificação em relação a TBox específica. O Axioma 2.11 é um exemplo de disjunção entre os conceitos C e D.

Tradicionalmente, o mecanismo básico implementado em motores de raciocínio DL é a subsunção de conceitos. A partir da subsunção é possível obter as demais inferências. Para isso são levados em consideração os conceitos superior e inferior.

Em relação a ABox a principal tarefa a ser utilizada é a **consistência** da ABox em relação a TBox implementada. Dessa forma uma ABox $\mathcal{A}$ é consistente em relação a TBox $\mathcal{T}$ se existe uma interpretação que é modelo de $\mathcal{A}$ e $\mathcal{T}$ ao mesmo tempo.

2.3 ONTOLOGIAS

2.3.1 Definições

Segundo Berners-Lee, Hendler e Lassila (2001) uma ontologia é um documento que define formalmente as relações a respeito de termos. Outra definição é que ontologias são esquemas de metadados que possui um vocabulário controlado de conceitos, onde cada um é explicitamente definido e sua semântica pode ser computacionalmente processada (MAEDCHE; STAAB, 2001). As ontologias são utilizadas como representações de conhecimento formais, algumas vezes restritos a um domínio particular (LEHMANN; VOELKER, 2014).

Outros autores também contribuíram com suas ideias acerca do conceito de ontologia, Shadbolt, Berners-Lee e Hall (2006) apresentam um conceito simplificado de ontologia ao dizer que as ontologias são formas e tentativas cautelosas de definir os dados e de permitir que os mesmos interajam com outros dados mantidos nos mais diferentes formatos.

2.3.2 RDF e RDF Schema

O RDF é um modelo de dados (ANTONIOU et al., 2012, Capítulo 3) que permite representar informações relacionadas aos dados utilizando triplas com sujeito(recurso), predicado (propriedade) e valor (declaração) (BERNERS-LEE; HENDLER; LASSILA, 2001). Uma propriedade é um atributo ou relação que descreve um recurso e uma declaração é um valor, podendo ser literal (numérico, texto, etc.) ou um outro recurso (KLEIN, 2001).

Um RDF pode ser facilmente mapeado como um grafo. Os recursos e literais são vértices do grafo (podendo diferenciá-los um do outro) e as propriedades são as ligações entre os recursos e literais (KLEIN, 2001).

O modelo RDF não define nada relacionada a sintaxe e por isso o XML é utilizado como uma de suas representações sintáticas, surgindo assim o RDF/XML, onde os elementos definidos no RDF são mapeados para os elementos do XML, sendo representados pelas *tags* do XML.

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Renan Fernandes</contact:fullName>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:mailbox rdf:resource="mailto:renan.fernandes@example.com"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.org/People/EM/contact#me">
    <rdf:type rdf:
        resource="http://www.w3.org/2000/10/swap/pim/contact#Person"/>
  </rdf:Description>
</rdf:RDF>
```

Código 2.1 – Exemplo de código RDF, contendo informações referentes ao contato de Renan Fernandes. No código RDF existem diversas triplas. A primeira possui como recurso a *Description*, como relação o nome completo e como valor o nome Renan Fernandes

No Código 2.1 é possível visualizar um arquivo RDF/XML que representa informações de contato da pessoa *Renan Fernandes*. Este exemplo utiliza-se de um *namespace* que já contém definições de elementos relacionados a elementos do RDF e dos contatos. Por exemplo a *tag rdf* : *Description* se refere a um elemento chamado *Description*, definido no endereço apresentado no atributo *xmlns:rdf*.

O RDF é um modelo de dados independente de domínio. Para que seja possível representar elementos de um determinado domínio é utilizado o RDF *Schema* (RDFS), que define o vocabulário utilizado no RDF (ANTONIOU et al., 2012, Capítulo 3). O RDFS possui alguns conceitos ontológicos como classes, propriedades, hierarquia e declaração de domínio e imagem para propriedades. Porém não possui alguns conceitos como equivalência, negação e disjunção (MCBRIDE, 2004). É possível utilizá-los em uma outra representação de conhecimento: OWL, que veremos a seguir.

2.3.3 OWL1 e OWL2

Para a codificação das ontologias na Web Semântica, a linguagem recomendada pelo W3C para construção de ontologias expressivas é a *Web Ontology Language* (OWL). A OWL foi construída como uma extensão do RDF e RDFS.

Em sua primeira versão, a OWL apresentou alguns problemas relacionados a limitações na expressividade, questões de sintaxe, meta-modelagem e questões relacionadas a versão (GRAU et al., 2008). Devido a esses problemas, foi desenvolvida a OWL 2 que é a versão atualmente recomendada pela W3C.

Uma característica que merece destaque sobre a OWL 2 é a existência de duas semânticas formais: OWL 2 *Full* e OWL 2 DL (ANTONIOU et al., 2012). A semântica OWL 2 *Full* permite que uma ontologia OWL 2 seja totalmente mapeada para RDF e com isso utilizar a RDF *Semantics*, semântica formal do RDF e RDFS. Já a OWL 2 DL é uma semântica diretamente relacionada a Lógica de Descrições e sua semântica formal é denominada *Direct Semantics*. Quanto ao tratamento computacional de ambas semânticas, a RDF *Semantics* é indecidível enquanto a *Direct Semantics* é decidível.

Por ter sido desenvolvida baseada em RDF e RDF *Schema*, a linguagem OWL 2 pode ser expressa usando a sintaxe RDF (ANTONIOU et al., 2012). Entretanto, além do OWL/XML, foram definidas algumas sintaxes adicionais com suas vantagens e desvantagens, dentre elas destacamos: *Funciontal-Style Syntax* e a *Manchester Syntax*. A seguir são descritas cada uma dessas sintaxes.

OWL/XML é uma sintaxe XML para a OWL 2, cujo benefício principal é permitir que desenvolvedores de ontologias possam manipulá-las utilizando ferramentas de edição de XML (ANTONIOU et al., 2012). Esse tipo de sintaxe possibilita o uso da sintaxe do RDF e RDF *Schema* combinado com a OWL 2. Considere o seguinte fato: "*Mãe é uma pessoa que tem algum filho*". Apresentamos a representação desse fato em OWL 2 a seguir.

```
<owl:Class rdf:about="http://www.cin.ufpe.br/~rlf5#Mae">
        <rdfs:subClassOf>
            <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                    <rdf:Description rdf:about="http://www.cin.ufpe.br/~rlf5#
                        Pessoa"/>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="http://www.cin.ufpe.br/~rlf5
                            #temFilho"/>
                        <owl:someValuesFrom rdf:resource="http://www.cin.ufpe.br/~
                            rlf5#Pessoa"/>
                    </owl:Restriction>
                </owl:intersectionOf>
            </owl:Class>
        </rdfs:subClassOf>
    </owl:Class>
```

Código 2.2 – Exemplo de código OWL 2

A sintaxe funcional (do inglês *Functional-Style Syntax*) é intimamente relacionada as estruturas formais de uma ontologia, sendo mais compacta e legível que outras sintaxes (ANTONIOU et al., 2012). A representação do fato "*Mãe é uma pessoa que tem algum filho*"em sintaxe funcional é *SubClassOf(Mae ObjectIntersectionOf(Pessoa ObjectSomeValuesFrom(temFilho Pessoa)))*.

A *Manchester Syntax* foi desenvolvida na Universidade de Manchester com foco em ser uma sintaxe legível por seres humanos tanto quanto possível (ANTONIOU et al., 2012). Diferentemente da sintaxe funcional em que os construtores são apresentados como funções, os construtores da *Manchester Syntax* são descritos com palavras em inglês correspondentes ao significado dos construtores OWL 2. O fato "*Mãe é uma pessoa que tem algum filho*"é representado em *Manchester Syntax* como *Mae SubClassOf: Pessoa and temFilho some Pessoa*.

Segundo as especificações sintáticas da OWL 2[5], as entidades são os blocos fundamentais das ontologias OWL 2. Algumas das entidades na OWL 2 são:

- **Classe**: conjunto de indivíduos. Está relacionada a um conceito da Lógica de Descrições;
- **Indivíduo**: representa objetos(instâncias) de um domínio;
- ***Object Property***: conecta pares de indivíduos. É responsável pelo relacionamento de um indivíduo com outro;
- ***Data Property***: conecta um indivíduo a algum tipo de dados (inteiro, texto, data, entre outros);

Uma ontologia OWL 2 possui como componente mais importante o seu conjunto de axiomas. Axiomas são declarações que indicam o que é verdade em um domínio. Como forma de obter axiomas mais complexos, uma ontologia na OWL 2 possui a capacidade de criar axiomas para classes e propriedades utilizando expressões anônimas, e não somente definições nomeadas (conceito A é subconceito do conceito B). Essas expressões anônimas, também conhecidas como *class expressions* podem ser obtidas através da combinação de construtores da OWL 2. Em "*Mae SubClassOf: Pessoa and temFilho some Pessoa*", representado na *Manchester Syntax*, onde *and* e *some* são construtores para *class expressions* com significado equivalente em DL a $\sqcap$ e $\exists$, respectivamente. Temos duas *class expressions* nesse exemplo anterior: *temFilho some Pessoa* é a *class expression* mais interna, utilizada na construção da *class expression mais externa*: *Pessoa and (temFilho some Pessoa)*.

Com o uso de uma semântica formal, as ontologias desenvolvidas em OWL 2 podem ser utilizadas por motores de raciocínio, adquirindo assim novos conhecimentos ou apresentando possíveis inconsistências. Dentre os motores de raciocínio disponíveis, destacam-se Pellet, HermiT, FaCT++ e Raccoon.

O `Pellet`[6] é um motor de inferência desenvolvido com o objetivo de implementar as funcionalidades necessárias para aplicações da Web Semântica (PARSIA; SIRIN, 2004). Dentre as funcionalidades disponíveis no `Pellet` é interessante destacar o suporte a dedução (*entailment*), que permite a verificação de satisfatibilidade e subsunção (PARSIA;

[5] https://www.w3.org/TR/owl2-syntax
[6] https://github.com/stardog-union/pellet/releases

SIRIN, 2004), além de ser possível também verificar inconsistências em conceitos presentes na ontologia, gerando *explanations* sobre estas.

O `HermiT`[7] é um raciocinador que utiliza o cálculo de *hypertableau*, diferentemente do `FaCT++` e `Pellet` que utilizam o cálculo de *tableau* (GLIMM et al., 2014). A utilização do *hypertableau* permite que o raciocinador evite alguns comportamentos não determinísticos do cálculo de *tableau*. Além do padrão estabelecido pela OWL 2, o `HermiT` suporta diversos serviços especialistas como regras *DL-safe* e consultas em SPARQL.

O raciocinador `FaCT++`[8] utiliza o cálculo de *tableau* e possui algumas técnicas de otimização padrão, além de ter desenvolvido algumas técnicas relacionadas a heurística para ordenação e classificação taxonômica (TSARKOV; HORROCKS, 2006).

O `Raccoon`[9] (FILHO; FREITAS; OTTEN, 2017) é um motor de raciocínio voltado a semântica OWL 2 DL e possui atualmente expressividade $\mathcal{ALC}$ e é baseado no método de conexões em DL $\mathcal{ALC}$ $\theta - CM$ (FREITAS; OTTEN, 2016).

2.4 REPRESENTAÇÃO VISUAL DE ONTOLOGIAS

Representação visual de dados é uma tarefa relevante para usuários que os manipulem, seja para entender de forma mais clara o que está representado ou para verificar se tudo está como o desejado.

No caso das ontologias, a visualização não é uma tarefa trivial. As ontologias não possuem somente relações de hierarquia como também possuem relações entre suas estruturas (conceitos, relacionamentos, instâncias) (KATIFORI et al., 2007).

A seguir são apresentadas algumas técnicas para a visualização de ontologias, descrevendo cada uma das técnicas e apresentando exemplos, e uma representação focada na semântica dos conceitos denominada *model outlines*.

2.4.1 *Idented List*

Técnica baseada na hierarquia dos conceitos da ontologia. Esta técnica é influenciada pelo método visual em que são apresentados os diretórios de um sistema operacional. Conceitos que possuem subconceitos são como diretórios, onde é possível expandir ou retrair seus elementos, e conceitos que não possuem subconceitos representados como arquivos. A principal vantagem desta técnica é a facilidade de uso, uma vez que a maioria dos usuários de computadores estão familiarizados com a técnica. Como desvantagem, a técnica não é capaz de representar a hierarquia múltipla, podendo o subconceito estar presente na árvore de um ou de todos os conceitos dos quais ele herda (KATIFORI et al., 2007). A ferramenta `Protégé` (MUSEN, 2015) utiliza esta técnica e representa a hierarquia múltipla inserindo o subconceito em todas as árvores que ele pertence.

[7] https://github.com/phillord/hermit-reasoner
[8] https://code.google.com/archive/p/factplusplus/
[9] https://github.com/dmfilho/raccoon/

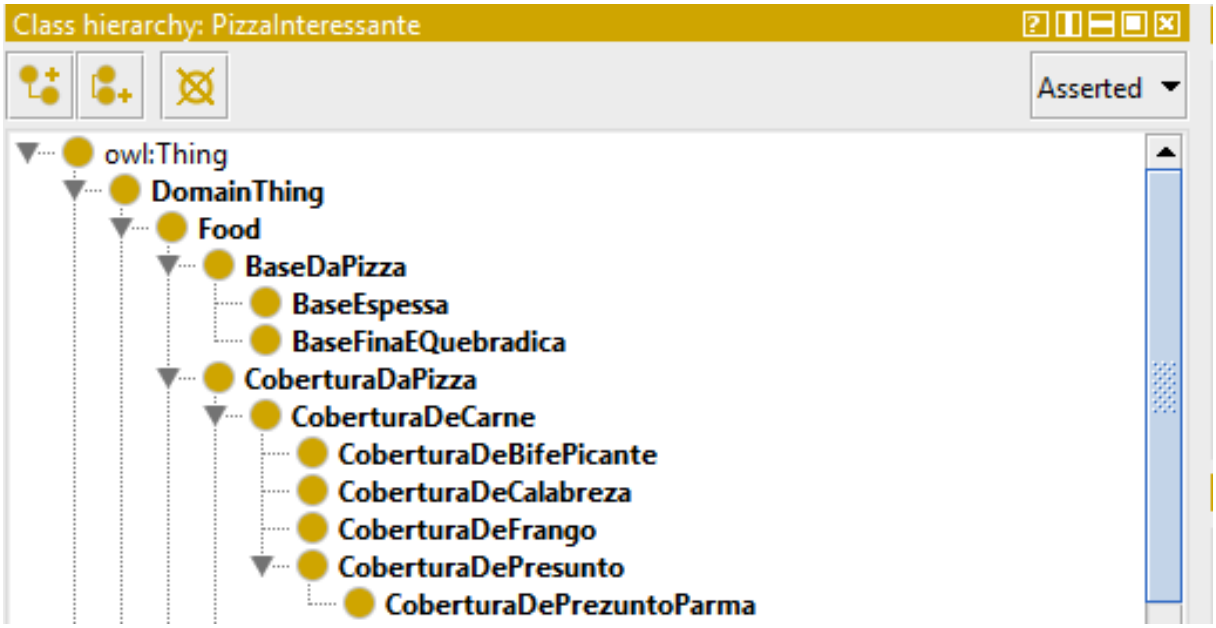


Figura 2 – Implementação da técnica *Idented List* na ferramenta `Protégé`.

Na Figura 2 é possível visualizar a implementação da técnica *Idented List* na ferramenta `Protégé`. A classe *owl:Thing* representa a classe superior da ontologia. E a partir dela é possível observar todas as outras classes pertencentes a hierarquia. Por exemplo, a classe *CoberturaDeFrango* é subclasse de *CoberturaDeCarne* que é subclasse de *CoberturaDaPizza* e assim sucessivamente. É importante comentar que a técnica de visualização *Idented List* pode causar confusão na classificação de classes que não estejam estritamente na hierarquia. Ou seja, em alguns casos uma classe pode ser vista em duas árvores diferentes, já que a sintaxe do OWL permite a hierarquia múltipla.

2.4.2 *Node-link and tree*

Diferentemente da *Idented List*, a técnica *Node-link and tree* representa bem ontologias onde seus conceitos possuem herança múltipla visto que se assemelha bastante a um grafo (KATIFORI et al., 2007). É uma técnica utilizada por diversas ferramentas e não existe uma padronização quanto aos componentes visuais. As classes por exemplo, na ferramenta *VOWL* (NEGRU; LOHMANN, 2013), são representadas como círculos, enquanto nas ferramentas *AVOnEd* (HALLAY et al., 2017), *Eddy* (LEMBO et al., 2016), *OntoTrack* (LIEBIG; NOPPENS, 2005) são representados como retângulos. Existem ainda outras ferramentas que fazem uma associação entre os elementos que compõem uma ontologia e o diagrama de classes da UML.

As desvantagens da técnica *Node-link and tree* envolvem o mau uso do espaço disponível já que sempre é possível ver espaços em branco ao utilizar esta técnica e a representação caótica que pode acontecer quando são representados centenas de nós ao mesmo tempo (KATIFORI et al., 2007).

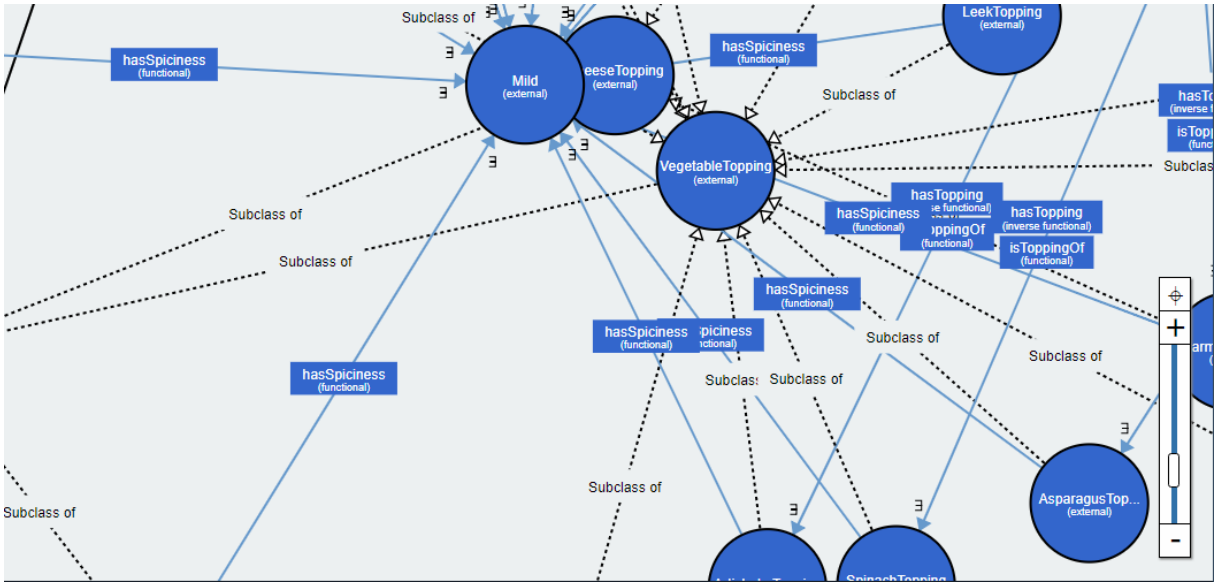


Figura 3 – Implementação da técnica *Node-link and tree* na ferramenta VOWL.

Algumas ferramentas utilizam abordagens para mitigar as desvantagens inerentes da técnica *Node-link and tree*. Ocultar ou manter tamanhos dinâmicos dos elementos visuais são abordagens úteis nesses casos. A ferramenta de representação visual `VOWL` utiliza a abordagem de tamanho dinâmico, alterando o tamanho dos elementos visuais de acordo com a sua utilização. Na Figura 3 é possível visualizar um fragmento da ontologia Pizza[10] representado na `VOWL`.

2.4.3 *Zoomable*

Técnica de visualização em que a hierarquia de classes é representada como conjuntos: o subconceito está contido no elemento que representa o conceito-pai. Ao selecionar um determinado subconceito da visualização, este passa então a ser o elemento-pai e é possível ver então os seus subconceitos. Com a técnica *Zoomable*, a visualização de toda a hierarquia da ontologia não é efetiva (KATIFORI et al., 2007). A ferramenta `OWLeasyViz` (CATENAZZI; SOMMARUGA; MAZZA, 2009) utiliza o tipo de técnica *Zoomable* para representar a hierarquia de classes da ontologia.

Na Figura 4 é possível visualizar a técnica *Zoomable* aplicada na ferramenta `OWLeasyViz`. As subclasses da classe *Processo*, como por exemplo *Gasometro* é representada, semelhante aos conjuntos, internamente a classe-pai. Caso deseje visualizar outras camadas da hierarquia de classe é necessário navegar entre as subclasses, gerando assim uma nova representação visual.

[10] https://protege.stanford.edu/ontologies/pizza/pizza.owl

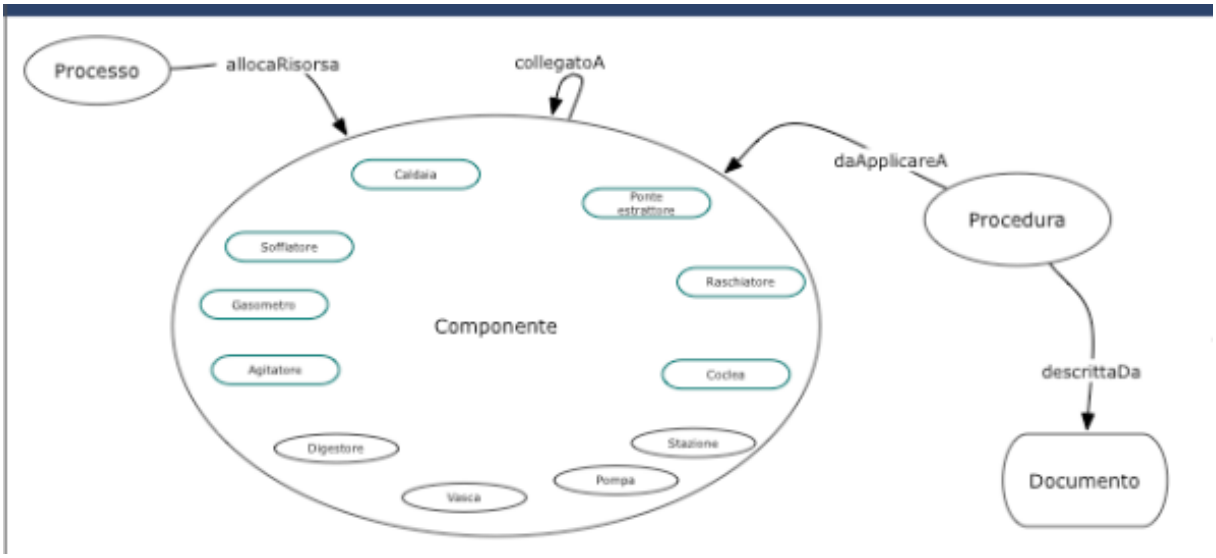


Figura 4 – Implementação da técnica *Zoomable* na ferramenta `OWLeasyViz`. Fonte: Catenazzi, Sommaruga e Mazza (2009)

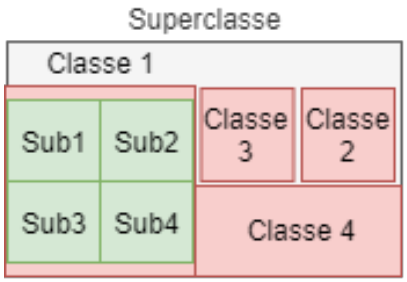


Figura 5 – Demonstração da técnica *Space-Filling*.

2.4.4 *Space-filling*

É uma técnica que busca utilizar todo o espaço da visualização, dividindo-o de acordo com propriedades dos vértices associados. Esta técnica é útil quando deseja-se visualizar informações dos nós-folha da árvore (KATIFORI et al., 2007).

Na Figura 5 é possível visualizar uma demonstração da técnica *Space-Filling*. Diferentemente da *Zoomable*, todas as possíveis subclasses da hierarquia já são apresentadas ao usuário internamente das superclasses. Na figura a classe *Sub1*, por exemplo, é subclasse de *Classe 1* que é subclasse da classe *Superclasse*.

A principal desvantagem dessa técnica é a dificuldade em representar os relacionamentos presentes nas ontologias. Além disso, deve-se existir diversas adaptações para suporte a visualização de indivíduos, por exemplo. Como vantagem, a técnica *Space-Filling* proporciona uma visualização de hierarquias que se assemelha a visualização de conjuntos.

2.4.5 Qual técnica utilizar?

Ainda segundo o trabalho de Katifori et al. (2007), nenhuma das técnicas vistas é a mais adequada para um propósito geral, depende da necessidade do usuário e, portanto, uma

possível solução é disponibilizar para o usuário diversas formas de visualização.

Na literatura, foi realizado um estudo controlado a respeito do mapeamento visual das classes de algumas ontologias, mostrando que *Idented Tree* (técnica equivalente a *Idented List*) é uma técnica organizada e familiar para novos usuários, além de ser mais efetiva para visualizar ontologias já mapeadas enquanto a técnica *Graph* (técnica equivalente a *Node-link and tree*) é mais controlável e intuitiva para os usuários, sem redundância e mais efetiva ao se utilizar ontologias que necessitem de alterações em seu mapeamento visual (FU; NOY; STOREY, 2013).

2.4.6 *Model Outlines*

Segundo Amaral (2008), muitos dos sistemas de visualização de descrições de conceito — termo utilizado em DL cujo significado equivale às *class expressions* do OWL — estão relacionados a sintaxe da linguagem que está sendo representada (DL e OWL, por exemplo), característica que pode impedir a compreensão dos usuários sobre a semântica dos conceitos visualizados. Nesse contexto, foi desenvolvida uma linguagem visual com foco maior na semântica dos construtores utilizados nas descrições de conceito denominada *model outlines* (AMARAL, 2008).

Considere as seguintes descrições de conceito, representadas em DL:

$$Pizza \sqcap \exists hasTopping.Mozzarella \sqcap \exists hasTopping.Tomato \tag{2.12}$$

$$Pizza \sqcap \forall hasTopping.(Mozzarella \sqcup Tomato) \tag{2.13}$$

$$\begin{aligned} Pizza \sqcap \exists hasTopping.Mozzarella \sqcap \exists hasTopping.Tomato \\ \sqcap \forall hasTopping.(Mozzarella \sqcup Tomato) \end{aligned} \tag{2.14}$$

De acordo com a semântica DL, a Descrição de Conceito 2.12 define os indivíduos que são pizza e que possuem cobertura de muçarela e cobertura de tomate. Entretanto, a descrição não restringe as relações de cobertura, podendo indivíduos pertencentes a esse conceito terem relações do tipo ter cobertura com indivíduos de outra classe. *Model Outlines* representa essa possibilidade de outros relacionamentos de cobertura com losangos pontilhados na Figura 6.

A Descrição de Conceito 2.13 define os indivíduos que são pizza e todas as relações do tipo ter cobertura devem ser realizadas com indivíduos que sejam tomate ou muçarela. Neste caso, a Descrição de Conceito 2.13 não obriga os indivíduos a terem pelo menos uma relação de ter cobertura. A não obrigatoriedade da relação ter cobertura é representada nos *model outlines* através de losangos pontilhados, podendo sua representação ser vista na Figura 7.

Se é desejado construir uma descrição de conceito que defina os indivíduos que são pizza e que possuem cobertura de muçarela e cobertura de tomate e nenhum outro tipo de indivíduo, deve-se utilizar a Descrição de Conceito 2.14. Nesta descrição de conceito, fica

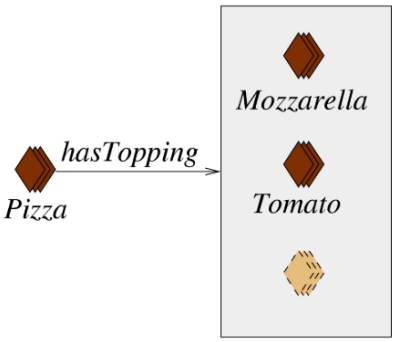


Figura 6 – Representação da linguagem visual *model outlines* para a Descrição de Conceito 2.12. Fonte: Amaral (2008)

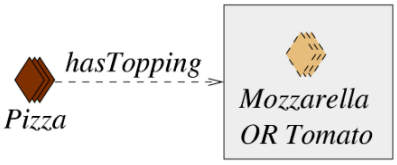


Figura 7 – Representação da linguagem visual *model outlines* para a Descrição de Conceito 2.13. Fonte: Amaral (2008)

claro, do ponto de vista da semântica da Lógica de Descrições, que os indivíduos imagem da relação ter cobertura devem ser tomate ou muçarela e nada diferente disso. A Figura 8 possui a representação desta descrição de conceito.

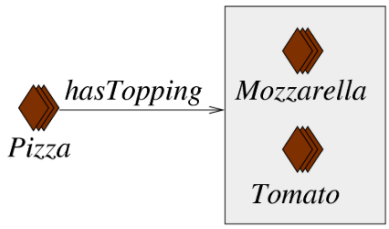


Figura 8 – Representação da linguagem visual *model outlines* para a Descrição de Conceito 2.14. Fonte: Amaral (2008)

3 MEDINA

O propósito principal desta dissertação é desenvolver um sistema de modelagem visual de ontologias e consequentemente demonstrar que esse sistema, por meio de interações com usuários, é capaz de construir uma ontologia com expressividade máxima da DL em $\mathcal{ALCHIQ}$ e realizar raciocínios de subsunção, detectar inconsistências, apresentando também visualmente as consequências e *explanations* desses raciocínios.

Tendo em vista que a construção de ontologias é também feita por quem não possui conhecimento em lógica, há uma necessidade real de ferramentas de autoria que forneçam interfaces intuitivas para construções de ontologias. A ferramenta aqui proposta utiliza em sua arquitetura, motores de raciocínio, APIs para manipulação de código OWL DL 2 e bibliotecas que auxiliem na construção da modelagem visual. Nas seções a seguir são apresentados a Arquitetura e seus componentes, exemplos de construção de ontologias utilizando o `Medina` e os algoritmos mais importantes desenvolvidos para uso no sistema proposto.

3.1 ARQUITETURA

A arquitetura do `Medina` pode ser dividida em quatro componentes principais, conforme apresentado na Figura 9. O componente GUI (1) (*Graphical User Interface*, Interface gráfica do usuário) é a interface de comunicação entre os usuários e a ferramenta. Os componentes *Medina Graph* (2) e *Ontology Core* (3) são os componentes responsáveis pelo gerenciamento da ontologia construída de forma visual por usuários e do código em OWL 2, respectivamente. O componente *Reasoner* (4) é responsável pelo raciocínio automático, realizado em tempo real durante a construção das ontologias.

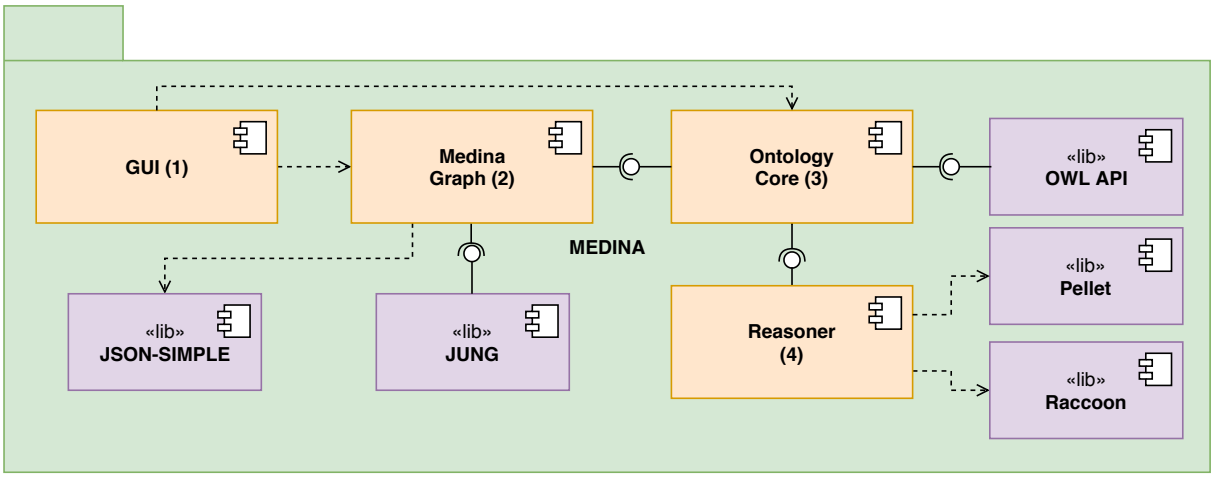


Figura 9 – Arquitetura do `Medina`, dividida entre seus principais componentes.

3.1.1 GUI

O componente GUI (1) apresentado na Figura 9 é o responsável pela comunicação dos usuários com os demais componentes do sistema proposto. A partir do GUI (1) o usuário cria, realiza manutenção nas ontologias criadas, importa e exporta ontologias existentes. O raciocínio automático ocorrido durante a construção das ontologias através da GUI também é apresentado de forma visual aos seus usuários e em tempo real.

De maneira técnica, o componente GUI (1) recebe como entrada ações realizadas pelos usuários, são exemplos de ações a importação de ontologias, criação de classes, de propriedades, de indivíduos e de relacionamentos entre classes, e as encaminha para o componente *Medina Graph* (2) ou o componente *Ontology Core* (3). Caso sejam alterações visuais, como a criação de classes, de propriedades, de indivíduos e de relacionamentos, são encaminhadas para o *Medina Graph*. Caso as ações sejam diretamente relacionadas ao código da ontologia como importação ou exportação de ontologias já desenvolvidas, então são encaminhadas para o *Ontology Core*.

3.1.2 *Medina Graph*

Sempre que alguma manipulação na ontologia é feita, é realizada uma operação equivalente no *Medina Graph* (2), componente 2 da arquitetura (Vide Figura 9). O componente *Medina Graph* (2) utiliza-se de uma estrutura de grafo direcionado para representar a ontologia. Classes, propriedades, indivíduos e construtores de classe são representados como vértices do grafo e axiomas são representados como arcos do grafo.

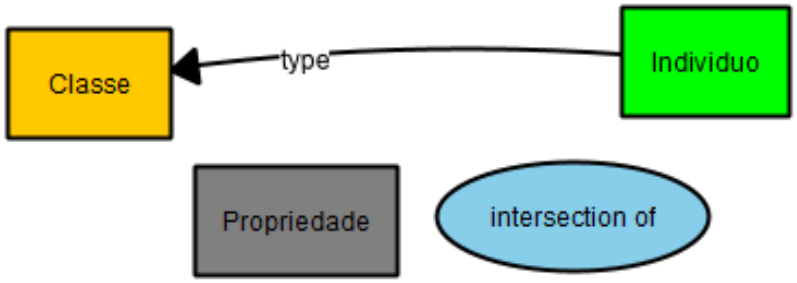


Figura 10 – Exemplo da representação visual em grafo dos elementos de uma ontologia no `Medina`

A representação visual em grafo dos elementos de uma ontologia no `Medina` é observada na Figura 10. As classes são representadas como retângulos na cor amarelo, as propriedades são representadas como retângulos na cor cinza, os indivíduos são representados como retângulos na cor verde e os construtores de classes representados como elipses na cor azul. É possível também observar na Figura 10 um axioma de asserção de classes, representando que o indivíduo *Indivíduo* é da classe *Classe*. A seguir são descritas as tecnologias utilizadas no componente *Medina Graph* (2).

3.1.2.1 Tecnologias

Para o gerenciamento e manipulação do grafo direcionado neste componente, foi utilizado o *framework* JUNG (O'MADADHAIN et al., 2005).

Java Universal Network Graph Framework[1] (Rede Universal de Grafos em Java) é uma biblioteca escrita em Java para modelar, analisar e visualizar dados que podem ser representados como grafos ou redes (O'MADADHAIN et al., 2005). Combinado com componentes da biblioteca *Swing* (API nativa da linguagem Java para interfaces gráficas), o JUNG permite que a modelagem dos grafos seja feita através de uma interface gráfica, simplificando a manipulação dos grafos e permitindo que outros desenvolvedores possam utilizar essa interface gráfica nos mais diversos projetos.

A utilização do JUNG é feita através das classes Java já implementadas ou de novas classes que referenciem interfaces do JUNG, permitindo a sua utilização nas mais diversas aplicações que necessitem trabalhar com grafos ou redes.

Uma funcionalidade interessante da ferramenta é a possibilidade de se utilizar diversos *layouts*, para que o grafo visualizado na interface gráfica possa ter seus vértices e arcos posicionados na tela automaticamente de acordo com o *layout* escolhido, trazendo a possibilidade de organizar grafos de diversas maneiras.

Por exemplo, considere o seguinte axioma em DL:

$$Class1 \sqsubseteq \forall prop.(Class2 \sqcup (Class3 \sqcap Class4))$$

É possível visualizar o axioma acima, organizado de maneira automática através de um dos *layouts* disponíveis na API JUNG, na Figura 11.

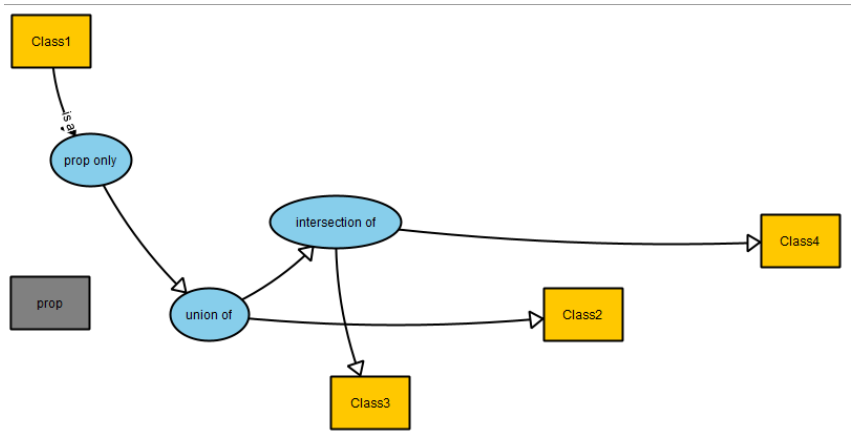


Figura 11 – Representação visual de uma ontologia utilizando um *layout* da API JUNG

Além do uso do JUNG para a manipulação de grafos no Medina, o componente *Medina Graph* (2) utiliza o `json-simple` para exportar e importar arquivos com informações visuais.

[1] https://sourceforge.net/projects/jung/files/

Para possibilitar a leitura e escrita de ontologias criadas no `Medina`, foram disponibilizadas funcionalidades relacionadas a escrita e leitura de arquivos com informações referente a posição dos nós e a conexão entre os nós. A extensão desse arquivo foi denominada DR e seu formato interno é o JSON[2] (*JavaScript Object Notation*, Notação de Objetos JavaScript).

JSON é uma sintaxe leve, independente de linguagem e baseada em texto para definir formatos de intercâmbio de dados. Para manipulação de arquivos em JSON, foi acoplado ao `Medina` a biblioteca `json-simple`.

`Json-simple` [3] é uma biblioteca implementada na linguagem Java que permite a abstração para desenvolvedores em relação a utilização de arquivos JSON, utilizando as classes implementadas pela biblioteca.

Os arquivos em formato JSON, se assemelham aos arquivos XML como uma forma interessante de se armazenar dados, os estruturando de forma simples em arquivos que também podem ser compreendidos por seres humanos, e não só pelas máquinas. A principal vantagem do JSON em relação ao XML é sua notação mais compacta e enxuta.

Considerando a representação visual representada na Figura 11, o grafo observado é mapeado para o formato JSON, persistindo informações referente a posição dos vértices na tela, tipo do vértice e sua visibilidade. É possível ver a ontologia desenvolvida no `Medina` em formato JSON no Código 3.1.

```
[{"visible":true,"x":46.27100785998154,"name":"Class1","y"
    :40.818468557294224,"isNode":true,"id":4,"type":"CLASS"},
{"visible":true,"x":552.5568100551774,"name":"Class2","y"
    :301.4458474549922,"isNode":true,"id":5,"type":"CLASS"},
{"visible":true,"x":350.4732405300136,"name":"Class3","y"
    :384.8792074352941,"isNode":true,"id":6,"type":"CLASS"},
{"visible":true,"x":759.6754747927589,"name":"Class4","y"
    :230.65844533127105,"isNode":true,"id":7,"type":"CLASS"},
{"visible":true,"x":44.50970744440796,"name":"prop","y"
    :290.2928090588129,"isNode":true,"id":8,"type":"OBJECT_PROPERTY"
    },
{"visible":true,"x":317.0046715730373,"name":"intersection of","y"
    :212.69706137813324,"isNode":true,"id":9,"type":"INTERSECTION"},
{"visible":true,"x":196.9876776376935,"name":"union of","y"
    :300.5037484737097,"isNode":true,"id":10,"type":"UNION"},
{"visible":true,"x":83.75244490345243,"name":"prop","y"
    :154.3532681797629,"isNode":true,"id":11,"type":"ONLY"},
{"sourceId":10,"name":"","axiomEdge":false,"isNode":false,"type":"
    CONNECTION","destinationId":5},
{"sourceId":4,"name":"is a","axiomEdge":false,"isNode":false,"type":
    "SUBCLASS_OF","destinationId":11},
```

[2] https://www.json.org/
[3] https://code.google.com/archive/p/json-simple/downloads

```
{"sourceId":9,"name":"","axiomEdge":false,"isNode":false,"type":"
    CONNECTION","destinationId":6},
{"sourceId":9,"name":"","axiomEdge":false,"isNode":false,"type":"
    CONNECTION","destinationId":7},
{"sourceId":11,"name":"","axiomEdge":false,"isNode":false,"type":"
    CONNECTION","destinationId":10},
{"sourceId":10,"name":"","axiomEdge":false,"isNode":false,"type":"
    CONNECTION","destinationId":9}]
```

Código 3.1 – Arquivo JSON gerado pela ferramenta `Medina`

3.1.3 *Ontology Core*

A geração e manipulação do código OWL DL 2 é realizada pelo componente 3 da arquitetura: o *Ontology Core* (3). A entrada para esse componente é uma ontologia em sintaxe intermediária. A sintaxe intermediária representa as *class expressions* como árvores. Representando como uma árvore uma *class expression*, ou classe complexa, é possível obter uma notação semelhante a visual e ao mesmo tempo próxima da sintaxe OWL DL 2.

As classes nomeadas e propriedades, quando necessárias, são representadas apenas por seus nomes na sintaxe intermediária. Os construtores de classe, cada um com suas restrições de construção, são os nós das árvores. O construtor complemento, por exemplo, só pode ter um único filho, sendo uma *string* (classe nomeada) ou outro nó. Já o construtor de restrição existencial, necessita de pelo menos um filho propriedade para que a notação intermediária seja válida. Considerando o seguinte axioma em DL:

$$Class1 \sqsubseteq \forall prop.(Class2 \sqcup (Class3 \sqcap Class4))$$

Sua representação na sintaxe intermediária, em forma de árvore, pode ser vista na Figura 12. A *class expression* $\forall prop.(Class2 \sqcup (Class3 \sqcap Class4))$ pode ser observada na estrutura de uma árvore. O construtor de restrição universal ($\forall$) é a raiz da árvore, seus filhos são o nome da propriedade e a classe aplicada a restrição: *prop* e $(Class2 \sqcup (Class3 \sqcap Class4))$ respectivamente.

A partir da ontologia visual de entrada o componente *Ontology Core* (3) realiza a transformação para código OWL DL 2, que é a saída do componente. Para a manipulação, importação e exportação das ontologias, o *Ontology Core* (3) utiliza como subcomponente a `OWL API` (HORRIDGE; BECHHOFER, 2011).

3.1.3.1 Tecnologias

Para o desenvolvimento deste componente foi utilizada a `OWL API`[4] (HORRIDGE; BECHHOFER, 2011), API com funcionalidades de exportar e importar ontologias OWL 2. É uma

[4] https://sourceforge.net/projects/owlapi/files/

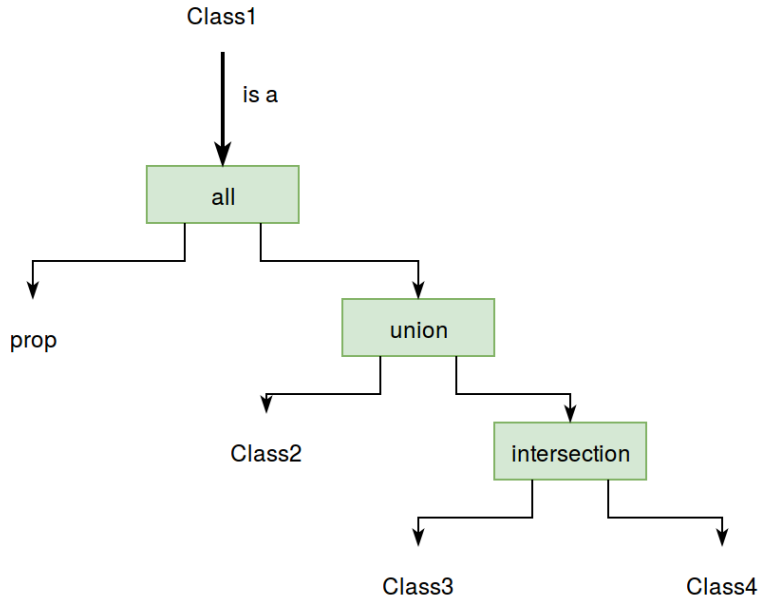


Figura 12 – Representação visual de um axioma na sintaxe intermediária no `Medina`

API desenvolvida na linguagem Java (DEITEL; DEITEL, 2011) para a geração e manipulação de ontologias em OWL. Ela suporta as especificações da OWL 2[5] e possibilita o uso combinado com motores de inferência. É utilizada em diversas ferramentas de manipulação de ontologias, como o `Protégé`[6] (MUSEN, 2015) e serviços online como conversores de sintaxe, validadores e repositórios de ontologias(HORRIDGE; BECHHOFER, 2011).

A `OWL API` permite que desenvolvedores possuam um considerável nível de abstração, desacoplando as funcionalidades específicas da ferramenta desenvolvida, das funcionalidades relacionadas ao gerenciamento das ontologias, como criação de classes e interfaces para a análise e geração de código OWL (HORRIDGE; BECHHOFER, 2011).

Para exemplificar a utilização da `OWL API` no `Medina`, considere o axioma representado na notação intermediária na Figura 12. É no componente *Ontology Core* que a sintaxe intermediária é convertida em OWL 2 efetivamente. No Código 3.2 é possível observar um trecho do código OWL 2, gerado pelo `Medina` através da `OWL API`.

```
<owl:Class rdf:about="http://www.cin.ufpe.br/~rlf5#Class1">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.cin.ufpe.br/~
                rlf5#prop"/>
            <owl:allValuesFrom>
                <owl:Class>
                    <owl:unionOf rdf:parseType="Collection">
                        <rdf:Description rdf:about="http://www.cin.
                            ufpe.br/~rlf5#Class2"/>
```

[5] https://www.w3.org/TR/owl2-syntax/
[6] http://http://protege.stanford.edu/

```
                                <owl:Class>
10                                  <owl:intersectionOf rdf:parseType="
                                        Collection">
                                        <rdf:Description rdf:about="http://
                                            www.cin.ufpe.br/~rlf5#Class3"/>
12                                      <rdf:Description rdf:about="http://
                                            www.cin.ufpe.br/~rlf5#Class4"/>
                                    </owl:intersectionOf>
14                              </owl:Class>
                            </owl:unionOf>
16                      </owl:Class>
                    </owl:allValuesFrom>
18              </owl:Restriction>
            </rdfs:subClassOf>
20      </owl:Class>
```

Código 3.2 – Exemplo de código OWL 2 gerado pelo `Medina`

3.1.4 *Reasoner*

O componente 4 da arquitetura (Vide Figura 9), é o *Reasoner* (4). Este componente é responsável pela comunicação em tempo real entre as ontologias manipuladas no *Ontology Core*(3) e os motores de raciocínio, mostrando possíveis insatisfatibilidade de classes, inconsistência da ontologia ou axiomas deduzidos por raciocínio de subsunção de maneira automática a cada nova alteração na ontologia em desenvolvimento.

O componente *Reasoner*, com o uso dos motores de raciocínio disponíveis, transfere as inferências e inconsistências obtidas, informações do axioma inferido ou da classe insatisfatível e o conjunto de axiomas que causam aquele raciocínio para o *Ontology Core*.

As tecnologias utilizadas neste componente são os motores de raciocínio `Pellet` e `Raccoon`, descritos na Seção 2.3.3, podendo ser extensível também para qualquer raciocinador que possua interface com a `OWL API`. Na subseção seguinte são apresentados os painéis disponíveis no `Medina`.

3.1.5 Painéis

A ferramenta `Medina` possui quatro painéis desenvolvidos para fornecer diversas funcionalidades para o usuário. Observamos na Figura 13 a tela principal do `Medina` e seus painéis 1, 2, 3 e 4.

A seguir são apresentados e descritos os painéis 1, 2, 3 e 4.

3.1.5.1 Painel 1 - Paleta de Ferramentas

O painel 1 (Vide Figura 13) é a paleta de ferramentas. Com essa paleta é possível construir visualmente os vértices e arcos necessários para iniciar o desenvolvimento de uma

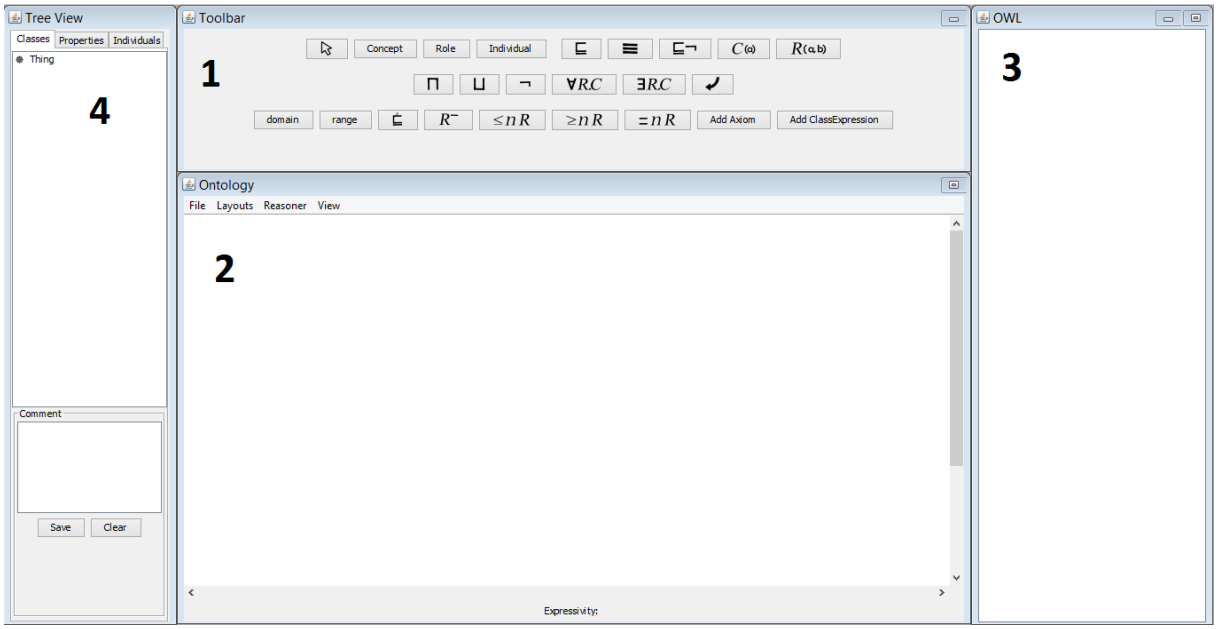


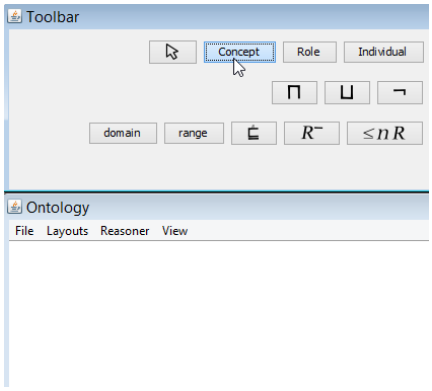
Figura 13 – Tela Principal da ferramenta.

ontologia. Na Tabela 3.1.5.1 é possível identificar a funcionalidade obtida em cada um dos botões da paleta de ferramentas.

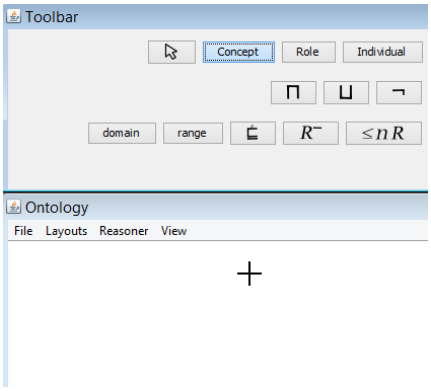
Tabela 2 – Ícones da paleta de ferramenta e suas funcionalidades

Ícone	Descrição	Exemplo
Mouse	Permite alterar a posição dos vértices da ontologia	
Concept	Novo conceito na ontologia	Pessoa
Role	Nova relação na ontologia	temCelular
Individual	Nova instância na ontologia	renan
$\sqsubseteq$	Novo axioma de subclasse	$Pessoa \sqsubseteq Animal$
$\equiv$	Novo axioma de equivalência	$HomoSapiens \equiv Pessoa$
$\sqsubseteq \neg$	Novo axioma de disjunção	$Animal \sqsubseteq \neg\ Pedra$
$C(a)$	Novo axioma de asserção de classe	$Pessoa(renan)$
$R(a, b)$	Novo axioma de asserção de relação	$temCelular(renan, celular1)$
$\sqcap$	Nova classe anônima do tipo interseção	$Pessoa \sqcap Macho$

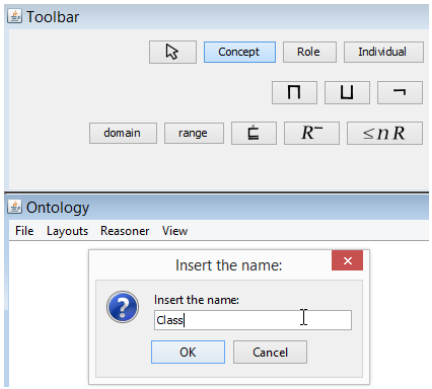
$\sqcup$	Nova classe anônima do tipo união	$Pessoa \sqcup Macho$
$\neg$	Nova classe anônima do tipo complemento	$\neg\ Macho$
$\exists r.C$	Nova classe anônima do tipo restrição existencial	$\exists\ temCelular.Celular$
$\forall r.C$	Nova classe anônima do tipo restrição universal	$\forall\ temCelular.Celular$
Seta	Novo arco do tipo conexão	
Domain	Novo axioma do tipo domínio, utilizado para indicar qual classe é a partida (domínio) de uma propriedade em questão	$\exists\ temCelular.\ \top \sqsubseteq Pessoa$
Range	Novo axioma do tipo imagem, utilizado para indicar qual classe é o destino (imagem) de uma propriedade em questão	$\top \sqsubseteq \forall\ temCelular.\ Celular$
$\overset{\text{r}}{\sqsubseteq}$	Novo axioma do tipo subpropriedade	$temPerna \sqsubseteq temMembro$
R^-	Novo axioma do tipo propriedade inversa	$ehCelularDe \equiv temCelular^-$
$\leq n\ r.C$	Nova classe anônima do tipo cardinalidade máxima	$\leq 2\ temCelular.\ Celular$
$\geq n\ r.C$	Nova classe anônima do tipo cardinalidade mínima	$\geq 1\ temCelular.\ Celular$
$=\ n\ r.C$	Nova classe anônima do tipo cardinalidade exata	$= 1\ temCelular.\ Celular$
Add axiom	Construção do axioma utilizando *Manchester Syntax*	*Pessoa EquivalentTo: HomoSapiens*
Add class expression	Construção de classe anônima utilizando a *Manchester Syntax*	*Pessoa and temCelular some Celular*



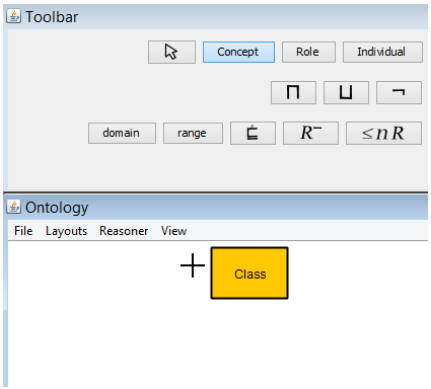
(a) Selecionando a opção no painel 1



(b) Escolhendo a posição no painel 2



(c) Inserindo o nome da classe no painel 2



(d) Classe adicionada no painel 2

Figura 14 – Exemplo de utilização dos painéis 1 e 2 na construção de ontologias no Medina

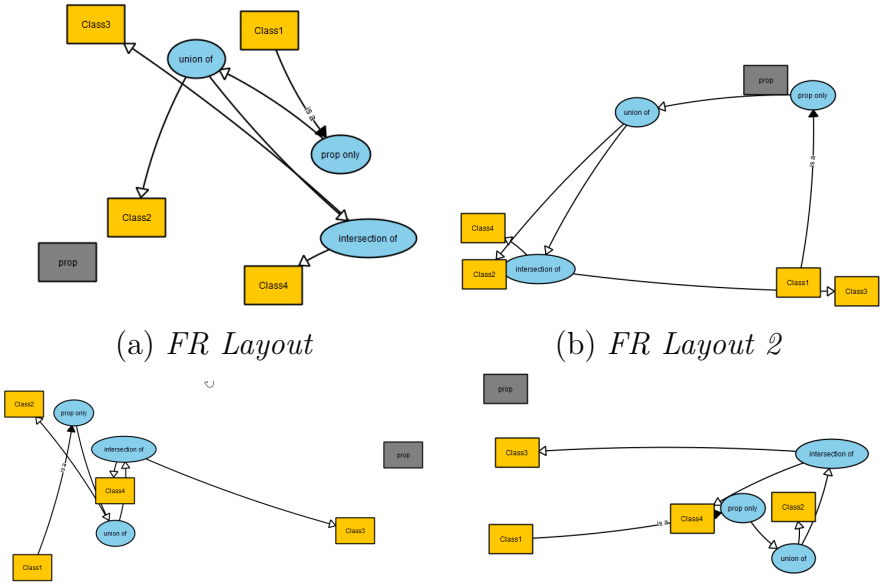
3.1.5.2 Painéis 2 e 3 - Construindo Ontologias

No painel 2 da Figura 13 o usuário constrói através de modelagem visual a ontologia através da interação com o painel 1. É no painel 2 que o usuário realiza a maior parte das interações no Medina.

Na Figura 14 é possível observar uma interação de exemplo, utilizando os painéis 1 e 2 para a construção de uma classe.

O painel 2 do Medina possui quatro menus. No menu *File* estão agrupadas as funcionalidades relacionadas a importar e exportar ontologias em OWL DL 2 ou DR, além de permitir exportar a representação visual como uma imagem. No menu *Layouts* o usuário pode escolher dentre os *layouts* disponíveis (através da interoperabilidade com o JUNG): *FR Layout*, *FR Layout 2*, *Spring Layout* e *ISOM Layout*. Cada um dos *layouts* possui uma implementação diferente, gerando uma representação visual diferente. Os quatro *layouts* disponíveis no Medina são vistos na Figura 15.

No menu *Reasoner* os usuários podem escolher qual motor de raciocínio utilizar entre as opções disponíveis. Até o momento os raciocinadores disponíveis são o Raccoon e o Pellet. O menu *View* proporciona ao usuário ocultar elementos visuais de acordo com seu tipo, sendo possível ocultar ou tornar visível nós dos tipos classe, propriedade, indivíduo e construtor de classe. No menu *View* também é possível ocultar axiomas inferidos, que



(a) *FR Layout* (b) *FR Layout 2*

(c) *Spring Layout* (d) *ISOM Layout*

Figura 15 – *Layouts* disponíveis no `Medina`.

são apresentados automaticamente para o usuário a cada manipulação na ontologia.

Em cada modelagem visual realizada pelo usuário é gerado um novo código contendo todas as entidades mapeadas da representação visual para código OWL correspondente. Um exemplo do código gerado no painel 3 pode ser visto na Figura 16.

OWL

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://www.cin.ufpe.br/~rlf5#"
     xml:base="http://www.cin.ufpe.br/~rlf5"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <owl:Ontology rdf:about="http://www.cin.ufpe.br/~rlf5"/>



    <!--
    ///////////////////////////////////////////////////////////////
    //
    // Classes
    //
    ///////////////////////////////////////////////////////////////
     -->




    <!-- http://www.cin.ufpe.br/~rlf5#Classe -->
```

Figura 16 – Código OWL 2 gerado e apresentado no Painel 3

3.1.5.3 Painel 4 - *Idented List*

O painel 4 (Vide Figura 13) pode ser utilizado pelos usuários por dois motivos. O primeiro é para uma visualização da hierarquia das classes ou propriedades utilizando a técnica de visualização *Idented List* (KATIFORI et al., 2007), em um estilo que se assemelha ao `Protégé` (MUSEN, 2015), onde a hierarquia de classes é apresentada como uma árvore de

diretórios de um sistema operacional. Na Figura 17 é possível observar um exemplo de uso do painel 4 para representação visual de uma hierarquia de classes.

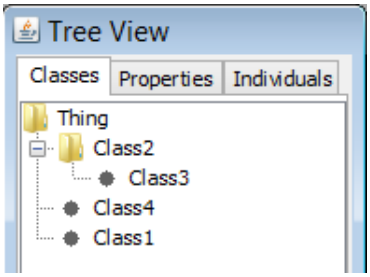


Figura 17 – Exemplo de representação visual do painel 4 no `Medina`.

A segunda motivação de uso do painel 4 é criar, remover ou editar comentários em linguagem natural sobre alguma das entidades da ontologia manipuladas. Os comentários inseridos no painel 4 são inseridos na ontologia com a tag do RDF *Schema rdfs:comment*.

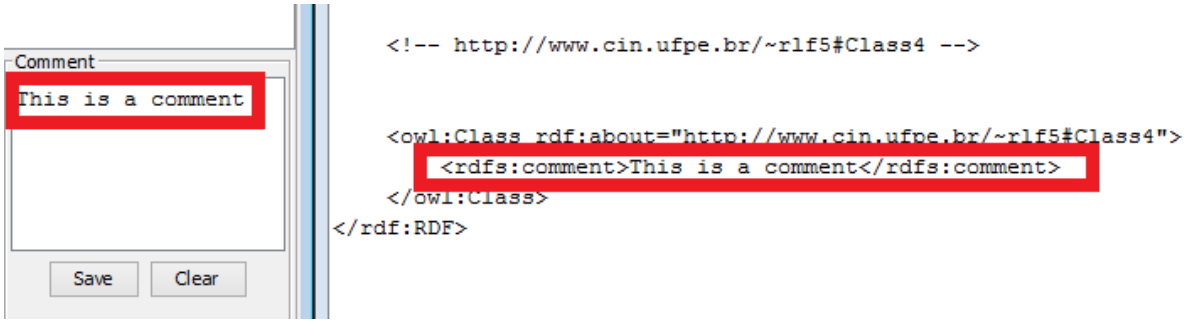


Figura 18 – Exemplo de adição de comentário no painel 4 do `Medina`.

Na Figura 18 é possível visualizar a adição de um comentário na classe *Class4* através do painel 4 e, consequentemente, o comentário foi adicionado à ontologia e apresentado no painel 3. O comentário está destacado em ambos os painéis dentro dos retângulos vermelhos.

3.2 COMPUTAÇÃO DA ONTOLOGIA ATRAVÉS DOS COMPONENTES

Apresentaremos a seguir, exemplos de uso das funcionalidades da Ferramenta `Medina`.

3.2.1 Importando uma ontologia

A primeira funcionalidade a ser apresentada é a de importação de uma ontologia. Seja a ontologia O sobre o domínio de Pizza e que possui os seguintes axiomas:

$$Pizza \sqsubseteq \forall hasTopping.Topping \tag{3.1}$$

$$Mozzarella \sqsubseteq Topping \tag{3.2}$$

O Axioma 3.1 define que todas as pizzas devem possuir como cobertura apenas indivíduos que sejam da classe *Topping*. Já o Axioma 3.2 define uma hierarquia de classes indicando que todos os indivíduos que forem da classe *Mozzarella* são também indivíduos da classe *Topping*.

Para importar a ontologia, assim como qualquer outra funcionalidade da ferramenta, o usuário deve utilizar a GUI (1) (Vide Figura 9). Ao acessar a funcionalidade visualmente de importar ontologia o componente GUI (1) terá como saída o diretório da ontologia O que deverá ser importada e passado como entrada para o *Ontology Core* (3) (Vide Figura 9). O Código 3.3 é um trecho da ontologia O, na linguagem OWL DL 2, a ser importada no `Medina`.

```
<owl:ObjectProperty
    rdf:about="http://cin.ufpe.br/~rlf5#hasTopping"/>

<owl:Class rdf:about="http://cin.ufpe.br/~rlf5#Mozzarela">
    <rdfs:subClassOf
        rdf:resource="http://cin.ufpe.br/~rlf5#Topping"/>
</owl:Class>

<owl:Class rdf:about="http://cin.ufpe.br/~rlf5#Pizza">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty
                rdf:resource="http://cin.ufpe.br/~rlf5#hasTopping"/>
            <owl:allValuesFrom
                rdf:resource="http://cin.ufpe.br/~rlf5#Topping"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="http://cin.ufpe.br/~rlf5#Topping"/>
```

Código 3.3 – Trecho de código OWL 2 referente a Ontologia O. Código gerado a partir dos Axiomas 3.1 e 3.2.

Ao receber como entrada o diretório da ontologia O, o *Ontology Core* (3) é responsável por importar todos os axiomas presentes na ontologia e encaminha os axiomas para os componentes *Medina Graph* (2) e *Reasoner* (4) (Vide Figura 9). Ao receber os axiomas, o componente *Reasoner* verifica possíveis raciocínios na ontologia O e os encaminha para o *Ontology Core* novamente. Neste exemplo não houveram raciocínios, portanto nenhuma informação é acrescentada.

O *Medina Graph* (2) realiza a construção do grafo direcionado de acordo com os axiomas recebidos como entrada. O axioma $Pizza \sqsubseteq \forall hasTopping.Topping$ possui um arco do tipo subclasse, saindo do vértice $Pizza$ em direção a um vértice *hasTopping only*, um vértice do tipo quantificador universal sobre a propriedade *hasTopping*.

Para que seja possível criar os construtores de restrição no `Medina`, é necessária a existência de uma propriedade declarada. Portanto, para o axioma $Pizza \sqsubseteq \forall\ hasTopping.\ Topping$, é criado um vértice do tipo propriedade, equivalente a propriedade da ontologia *hasTopping* e em seguida construído o vértice *hasTopping only*. Na Figura 19 é possível ver a importação da ontologia O até o momento.

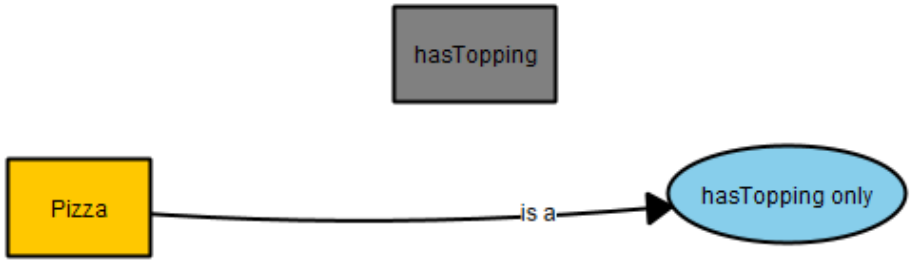


Figura 19 – Importando a ontologia O no `Medina`.

O vértice *hasTopping only* possui um arco do tipo conexão saindo em direção ao vértice $Topping$, terminando assim a representação em grafo do axioma $Pizza \sqsubseteq \forall\ hasTopping.\ Topping$. A importação da ontologia O até o momento é observada na Figura 20.

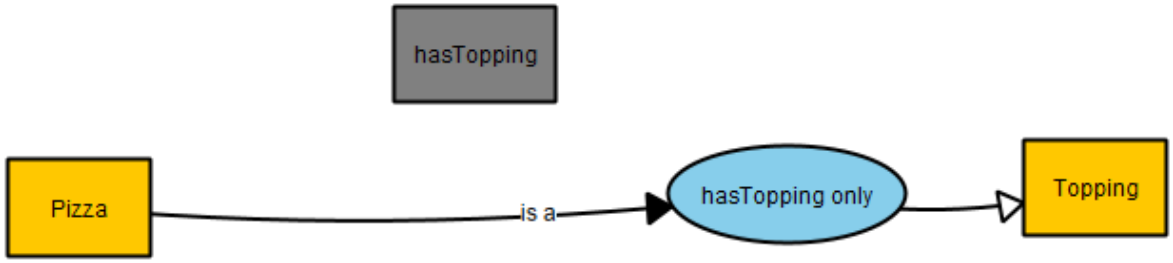


Figura 20 – Importando a ontologia O no `Medina`.

O axioma $Mozzarella \sqsubseteq Topping$ possui apenas um arco do tipo subclasse, saindo do vértice $Mozzarella$ em direção ao vértice $Topping$. Após a adição do vértice $Mozzarella$ e a ligação através de um arco de subclasse com a classe $Topping$ a ontologia O é completamente mapeada. Na Figura 21 é possível ver a ontologia O totalmente mapeada em representação visual.

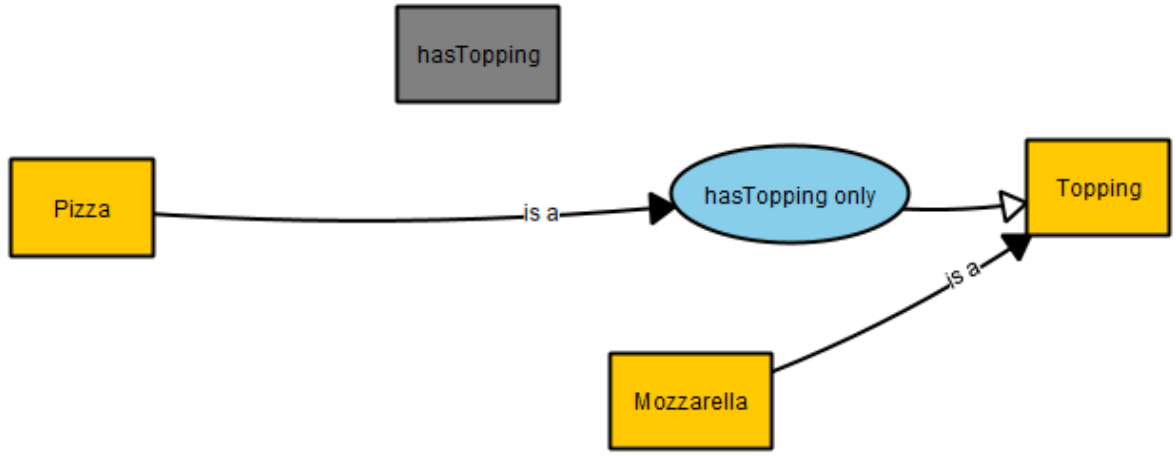


Figura 21 – Representação Visual da ontologia importada na computação. A ontologia O possui três conceitos ($Pizza$, $Topping$ e $Mozzarella$), e uma propriedade ($hasTopping$)

Com o mapeamento da ontologia O para um grafo direcionado finalizado, o componente *Medina Graph* (2) passa como saída o grafo mapeado para o componente GUI (1). Ao receber como entrada o grafo mapeado, o componente GUI (1) apresenta visualmente o grafo direcionado para o usuário.

3.2.2 Adicionando novos axiomas visualmente

Baseando-se ainda na ontologia O, outra computação relevante dos componentes do `Medina` é a de adicionar novos axiomas a ontologia em desenvolvimento. Neste caso, o componente GUI (1) recebe como entrada operações de adição de vértices ou arcos no grafo. As adições, consequentemente, são mapeadas para o grafo que representa a ontologia pelo componente *Medina Graph* e adicionadas ao código OWL DL 2, em tempo real, pelo componente *Ontology Core*. Após a geração do código OWL DL 2, é então realizado o raciocínio sobre a ontologia manipulada pelo componente *Reasoner* de maneira automática.

Para apresentar a adição de axiomas, considere o Axioma 3.3, representado em DL:

$$MozzarellaPizza \equiv Pizza \sqcap \exists hasTopping.Mozzarella \tag{3.3}$$

O Axioma 3.3 define que uma $MozzarellaPizza$ é uma $Pizza$ e que possui pelo menos uma cobertura ($hasTopping$) de $Mozzarella$. Reforçando que para a pizza ser

MozzarellaPizza nesse domínio ilustrativo, ela deve ter como cobertura no mínimo *Mozzarella*, mas de acordo com a semântica da DL, ela pode ter outras coberturas.

Para que o usuário possa adicionar o conceito *MozzarellaPizza*, ele utilizará a GUI (1), selecionando a opção de adição de conceito e em seguida clicando na posição onde deseja inserir o conceito no grafo (semelhante ao exemplo na Figura 14). Quando o componente GUI recebe como entrada a adição de um vértice, o componente *Medina Graph* (2) adiciona o vértice no grafo direcionado. A representação visual da adição do conceito à ontologia pode ser vista na Figura 22.

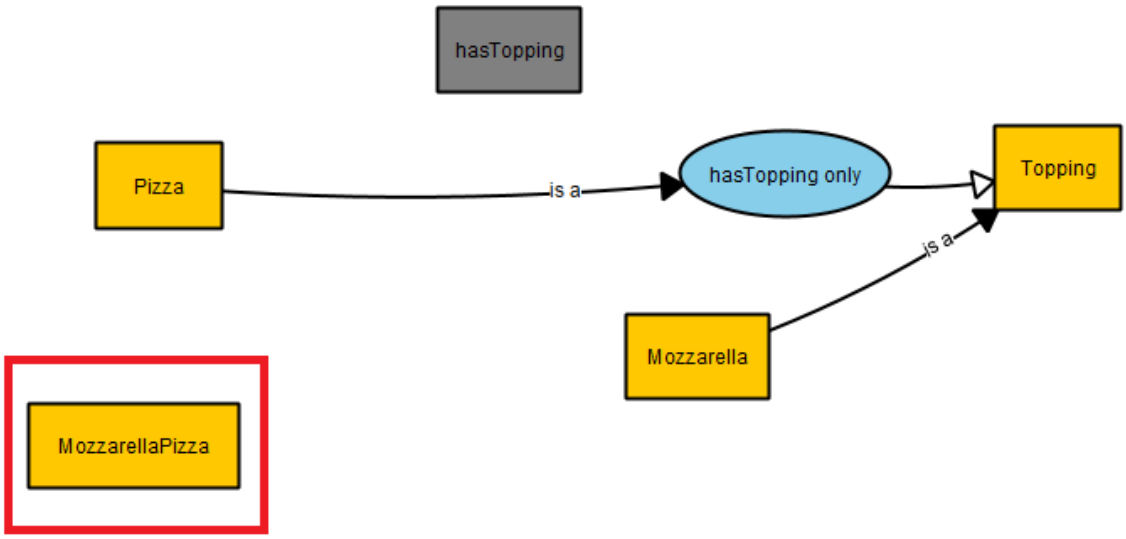


Figura 22 – Adição do conceito *MozzarellaPizza*, destacado com o retângulo vermelho.

O axioma $MozzarellaPizza \equiv Pizza \sqcap \exists\ hasTopping.\ Mozzarella$, possui dois tipos de construtores de classes anônimas: interseção ($\sqcap$) e restrição existencial ($\exists$). Os construtores são representados na ferramenta como vértices em que não é necessária a adição de nome. No caso da interseção basta selecionar visualmente a opção do construtor na paleta de ferramentas (Painel 1 da Figura 13) e clicar onde deseja adicionar o vértice. Na Figura 23 é possível observar a adição dos construtores de interseção e restrição existencial na ontologia.

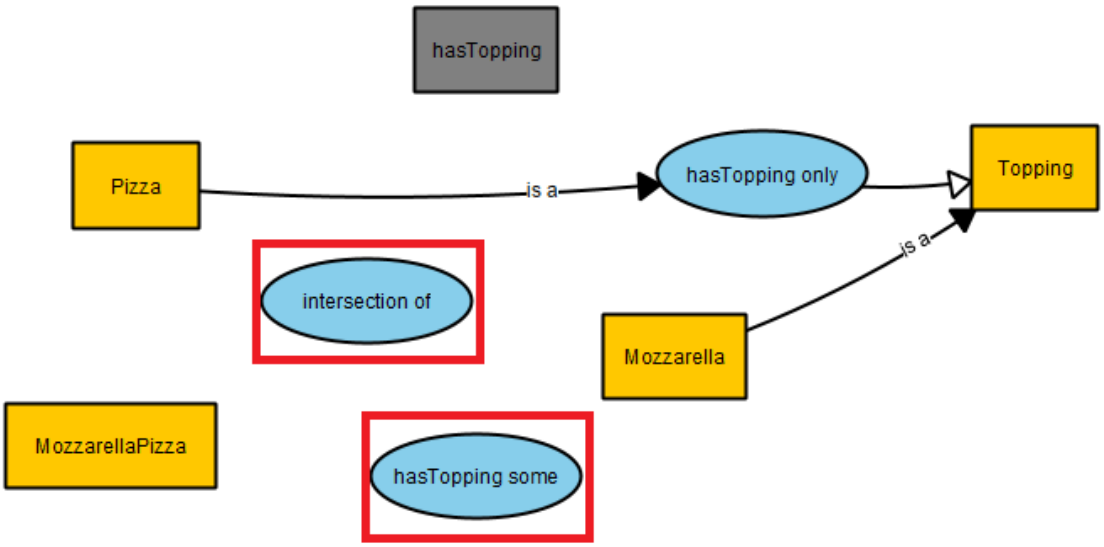


Figura 23 – Adição dos construtores de interseção e restrição existencial na ontologia destacados em retângulos vermelhos.

Criados todos os vértices necessários a próxima etapa é a criação de arcos ligando os vértices para assim formarem o axioma desejado. Para possibilitar a criação de classes anônimas complexas são utilizados arcos do tipo conexão. Arcos de conexão partem de vértices construtores de classe com destino a outros vértices de construtores de classe ou classes nomeadas.

O primeiro arco que deverá ser adicionado é entre o vértice de interseção e o vértice *Pizza*. Em seguida, são criados os arcos que ligam a interseção com o construtor de restrição e deste ao conceito *Mozzarella*. Por fim, é criado o arco de axioma de equivalência ligando os vértices de interseção e *MozzarellaPizza*. Na Figura 24 é possível observar os arcos de conexão (destacados em vermelho) e de equivalência (destacado em verde) adicionados a modelagem visual.

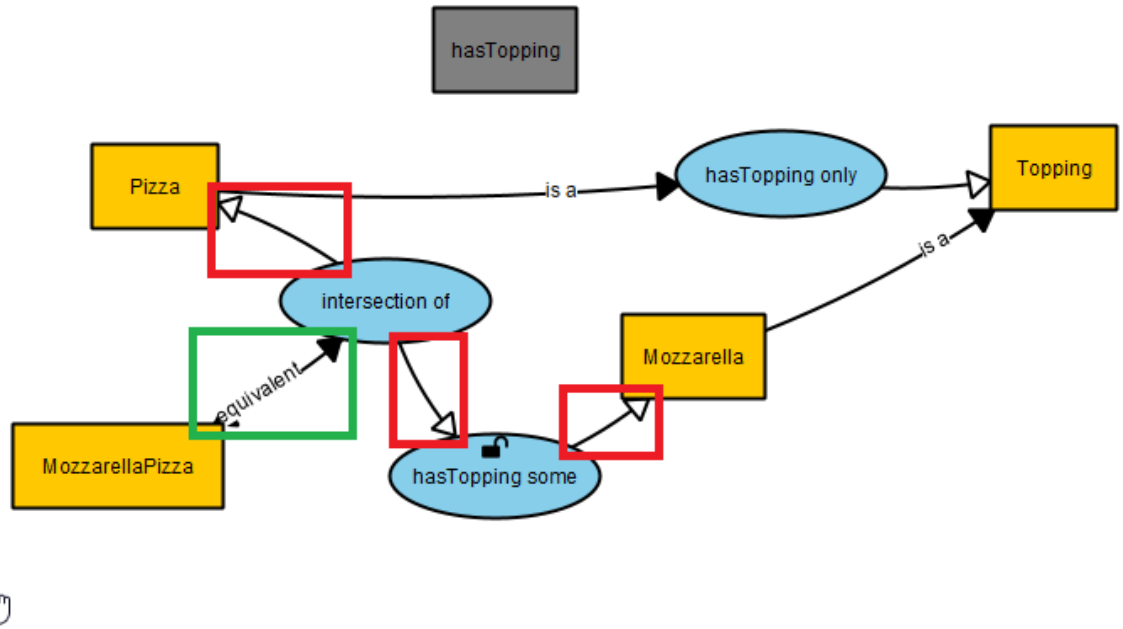


Figura 24 – Adição dos arcos de conexão e de equivalência. Os arcos de conexão estão destacados sob retângulos em vermelho. O arco de equivalência está destacado sob o retângulo verde.

Ao computar a ontologia, o componente *Reasoner* retorna um axioma deduzido a partir da ontologia.

$$MozzarellaPizza \sqsubseteq Pizza \tag{3.4}$$

Nessa parte da execução, o componente *Reasoner* encaminha a ontologia, acrescida da informação de dedução, para o componente *Ontology Core* e este, ao receber como entrada a dedução, encaminha para o *Medina Graph*, que irá adicionar um novo arco no grafo, ligando os conceitos de $MozzarellaPizza$ e $Pizza$. Por ser inferido, o arco não é adicionado ao código OWL da ontologia automaticamente. Sua representação visual é diferenciada, sendo os arcos de inferência representados como setas tracejadas. O componente *Medina Graph* repassa o grafo atualizado para a GUI que apresenta o grafo visualmente ao usuário. A representação visual do axioma pode ser vista na Figura 25 com o axioma inferido destacado em vermelho.

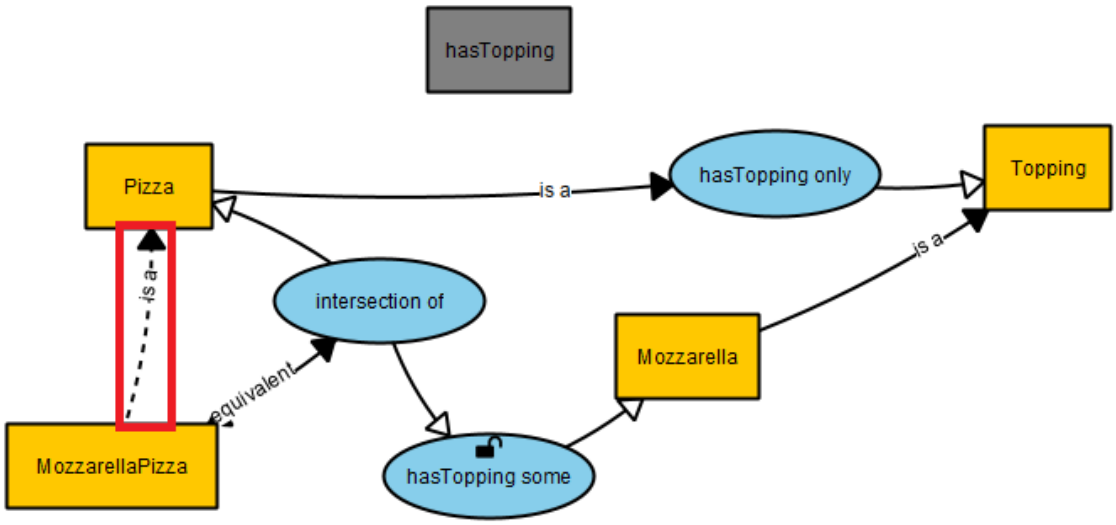


Figura 25 – Representação Visual da ontologia modelada, com destaque para o axioma deduzido (inserido no retângulo vermelho).

A representação do axioma $MozzarellaPizza \equiv Pizza \sqcap \exists hasTopping.Mozzarella$ em OWL DL 2, feita pelo *Ontology Core* (3), pode ser vista no Código 3.4.

```
<owl:Class rdf:about="http://cin.ufpe.br/~rlf5#MozzarellaPizza">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <rdf:Description
                    rdf:about="http://cin.ufpe.br/~rlf5#Pizza"/>
                <owl:Restriction>
                    <owl:onProperty
                        rdf:resource="http://cin.ufpe.br/~rlf5#hasTopping"/>
                    <owl:someValuesFrom
                        rdf:resource="http://cin.ufpe.br/~rlf5#Mozzarella"/>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
</owl:Class>
```

Código 3.4 – Código OWL DL 2 referente ao Axioma 3.3.

3.2.3 *Visual Explanation* de Inferências

Uma das principais funcionalidades da ferramenta é a *visual explanation* de inferências. O axioma $MozzarellaPizza \sqsubseteq Pizza$ foi inferido de acordo com a semântica da DL sobre o conjunto de axiomas inseridos na ontologia O. Quando o usuário passar o *mouse* sobre o axioma inferido são apresentadas *explanations* textuais e visuais para o usuário. Na Figura 26 é possível observar a *visual explanation* da inferência apresentada. Todos os elementos visuais que fazem parte dos axiomas da *visual explanation* mudam para a cor azul ciano, destacando para o usuário quais são os axiomas relacionados a inferência.

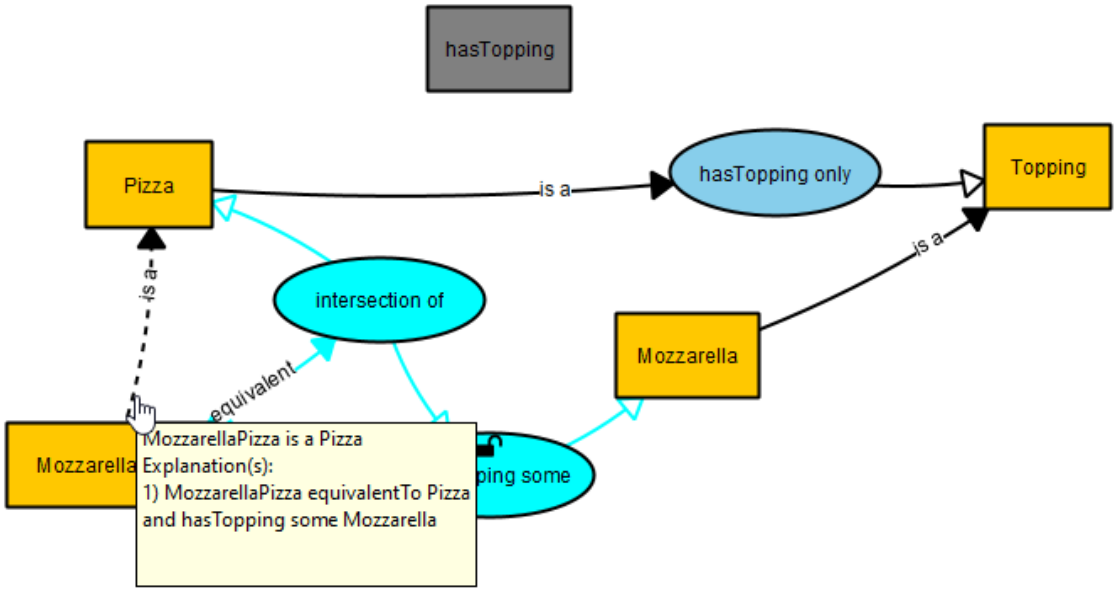


Figura 26 – *Visual Explanation* e Textual da Inferência obtida na ferramenta.

3.2.4 Fecho automático de axiomas

Para que alguns raciocínios sejam realizados de acordo com o esperado pelo usuário, é necessário a inserção de axiomas de fecho. No `Medina` está disponível a funcionalidade de adição de axiomas de fecho para propriedades de maneira automática. Considere o seguinte axioma:

$$MozzarellaPizza \equiv Pizza \sqcap \exists hasTopping.Mozzarella$$

Os indivíduos que são do tipo $MozzarellaPizza$ devem também ser do tipo $Pizza$ e possuir pelo menos uma relação $hasTopping$ com um indivíduo $Mozzarella$. Entretanto, esse axioma não restringe que $MozzarellaPizza$ possa ter coberturas de outros sabores. Isso ocorre devido a semântica de mundo aberto da ABox (descrita na Seção 2.2.2). Para que seja possível definir que os indivíduos devem ter apenas cobertura de $Mozzarella$, e de nenhum outro sabor, é necessário adicionar o axioma $MozzarellaPizza \sqsubseteq \forall\ hasTopping$.

$Mozzarella$ à ontologia. Neste axioma, de acordo com a semântica da DL, restringe que os indivíduos do tipo $MozzarellaPizza$ devem ter as relações $hasTopping$ apenas com indivíduos do tipo $Mozzarella$.

Os axiomas de fecho podem ser construídos no `Medina` a partir de construtores existenciais.

No caso do axioma $MozzarellaPizza \equiv Pizza \sqcap \exists\ hasTopping.\ Mozzarella$, caso o usuário deseje, é possível criar o axioma de fecho de maneira automática, clicando apenas no construtor que possui o cadeado aberto indicando a ausência do fecho[7]. Ao criar o fecho, é acrescido à ontologia o axioma $MozzarellaPizza \sqsubseteq \forall\ hasTopping.\ Mozzarella$ e a representação visual da ontologia com o axioma pode ser vista na Figura 27.

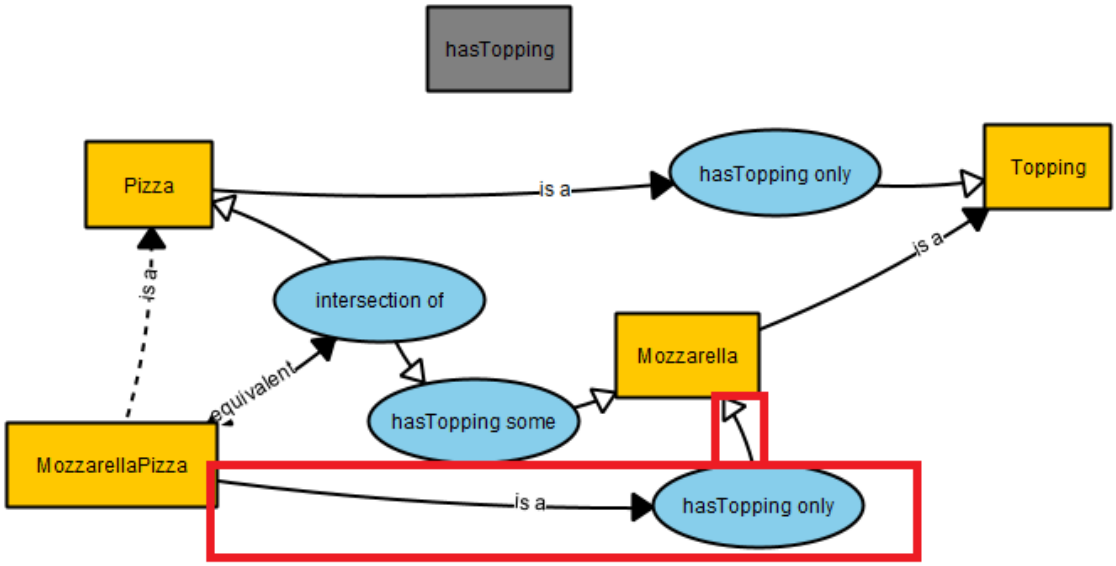


Figura 27 – Representação Visual da Ontologia O após a inserção do axioma de fecho. O axioma de fecho está destacado em dois retângulos vermelhos.

3.2.5 Verificando insatisfatibilidade de classes

Continuando com o exemplo da ontologia O, a fim de exemplificar a funcionalidade de raciocínio de insatisfatibilidade, poderia ser inserido um axioma de disjunção entre $Pizza$ e $MozzarellaPizza$ e sua representação em DL pode ser vista no Axioma 3.5.

$$Pizza \sqsubseteq \neg MozzarellaPizza \tag{3.5}$$

Ao adicionar o Axioma 3.5, o componente *Reasoner* (4) (Vide Figura 9) irá detectar que $MozzarellaPizza$ se tornou uma classe insatisfatível. De acordo com a semântica da DL (Vide Seção 2.2.2), todo indivíduo que é uma $MozzarellaPizza$ também é uma $Pizza$.

[7] O ícone de cadeado foi utilizado em alusão ao termo "fecho"do axioma

Todavia, com a adição do axioma $Pizza \sqsubseteq \neg MozzarellaPizza$, é inferido também que não podem existir indivíduos que sejam $Pizza$ e $MozzarellaPizza$. Ora, se todo indivíduo $MozzarellaPizza$ deve ser uma $Pizza$, porém não pode existir um indivíduo que seja ambos os conceitos, a classe $MozzarellaPizza$ não pode ter indivíduos, caracterizando a insatisfatibilidade.

Caso o usuário deseje verificar quais axiomas explicam esse raciocínio, basta passar o *mouse* sobre $MozzarellaPizza$. Na Figura 28 é possível observar a *explanation* textual e visual da insatisfatibilidade de $MozzarellaPizza$. Em vermelho, os axiomas que causaram a insatisfatibilidade de $MozzarellaPizza$, são destacados provendo um *feedback* para os usuários de que ali ocorreu uma insatisfatibilidade e esses são os axiomas que causaram o raciocínio.

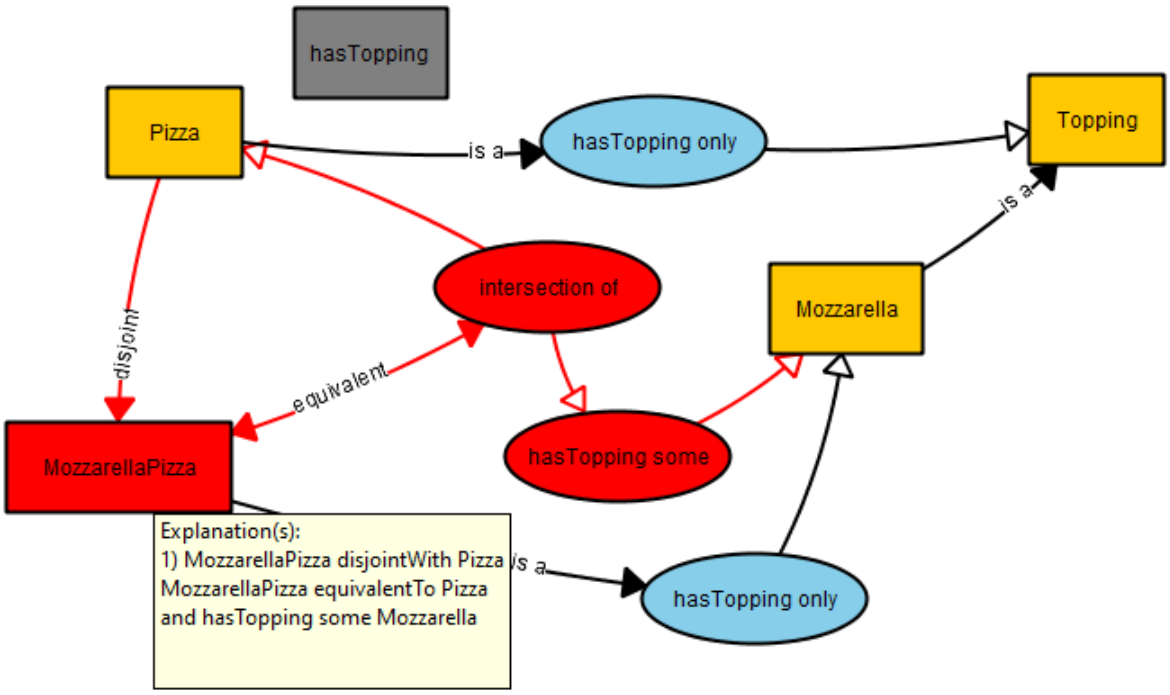


Figura 28 – *Visual explanation* de uma insatisfatibilidade obtida na ferramenta.

3.2.6 Ocultando axiomas complexos

Um dos maiores problemas na utilização de grafos para representação visual de maneira geral ocorre quando o grafo vai se tornando muito grande. A fim de mitigar esse problema, a ferramenta possui a funcionalidade de ocultar axiomas complexos.

Ao clicar com o botão direito no arco de equivalência do axioma $MozzarellaPizza \equiv Pizza \sqcap \exists\ hasTopping.\ Mozzarella$ (destacado na Figura 24), o usuário poderá ocultar esse axioma. Perceba que o axioma em questão possui a classe complexa $Pizza \sqcap \exists\ hasTopping.\ Mozzarella$ caracterizando assim um axioma complexo. Na Figura 29 é

possível observar a ontologia O e o ícone em $MozzarellaPizza$ representando o axioma ocultado.

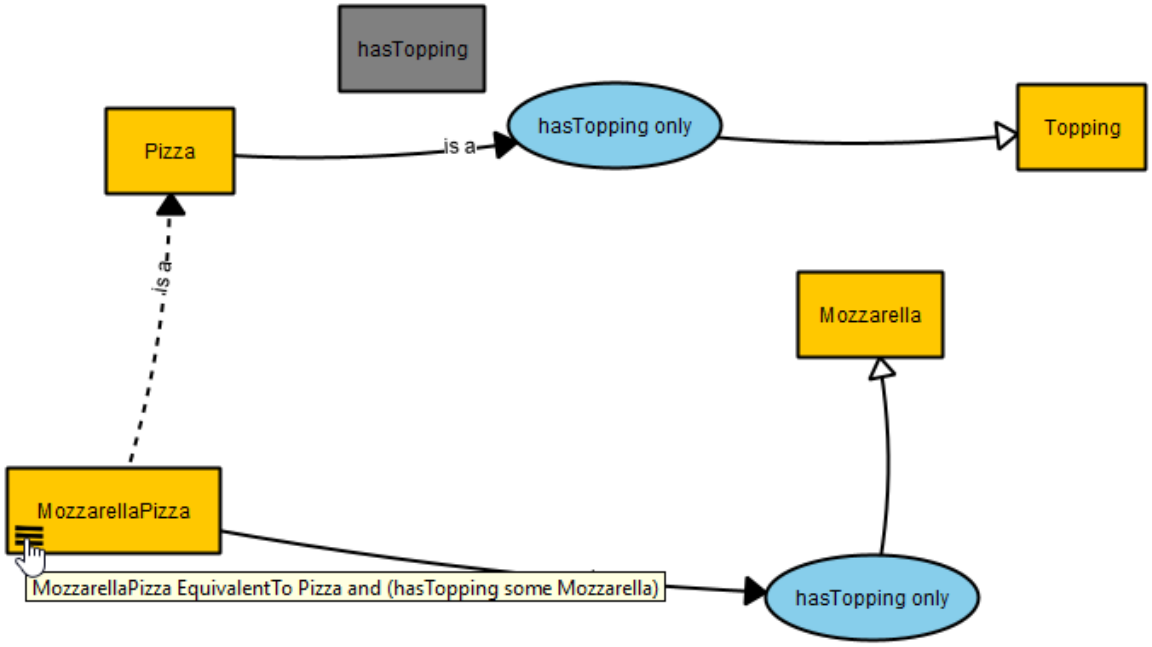


Figura 29 – A ontologia exemplo e o axioma agora oculto visualmente e representado com o símbolo de equivalência em $MozzarellaPizza$.

Caso o usuário deseje entender qual é o axioma oculto, passando o *mouse* por cima do símbolo de equivalência será mostrada uma descrição, em *Manchester Syntax*, do axioma ocultado, como representado na Figura 29. Clicando no símbolo será possível restaurar o axioma ocultado. Além da funcionalidade de ocultar axiomas, a ferramenta também proporciona ocultar classes, propriedades e indivíduos de maneira individual ou por tipo disponível no menu *View* do painel 2 (Vide Figura 13).

3.2.7 Inserindo axiomas e classes complexas com *Manchester Syntax*

A OWL 2 possui diferentes sintaxes para a representação de ontologias. Uma delas é a *Manchester Syntax* (Vide Seção 2.3.3 do Capítulo 2), desenvolvida para ser uma sintaxe mais próxima da linguagem humana. A adição de axiomas no `Protégé` permite o uso de *Manchester Syntax* ao representar *class expressions*.

Considere o seguinte axioma em DL:

$$MargheritaPizza \equiv Pizza \sqcap \exists hasTopping.Mozzarella \sqcap \exists hasTopping.Tomato \quad (3.6)$$

Para utilizar o Axioma 3.6 nesta funcionalidade é necessário utilizar a sua representação em *Manchester Syntax*. Nesse caso, os símbolos que representam a equivalência, interseção e restrição existencial são substituídos por *EquivalentTo:*, *and* e *some* respectivamente. Portanto, o Axioma 3.6 é representado em *Manchester Syntax* como *Marghe-*

ritaPizza EquivalentTo: Pizza and hasTopping some Mozzarella and hasTopping some Tomato.

Para adicionarmos esse axioma no *Medina* é necessário que todas as classes e propriedades utilizadas no axioma já estejam inseridas previamente na ferramenta. Após a adição das classes e propriedades utilizadas no axioma (Vide Figura 30), basta selecionar a opção *Add axiom* na paleta de ferramentas do Medina (Vide Figura 31). Será então apresentado ao usuário uma nova janela com um campo a ser preenchido pelo usuário digitando o axioma em *Manchester Syntax* (Vide Figura 32).

Figura 30 – Modelagem visual de classes e propriedades.

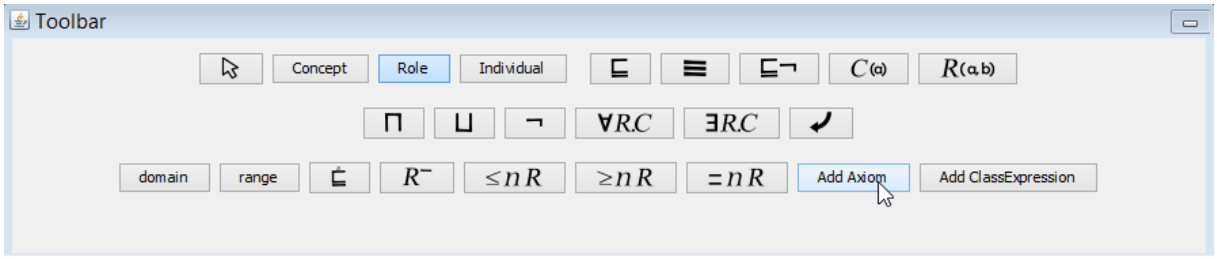


Figura 31 – Momento de escolha da opção de adição de axiomas em *Manchester Syntax* na paleta de ferramentas

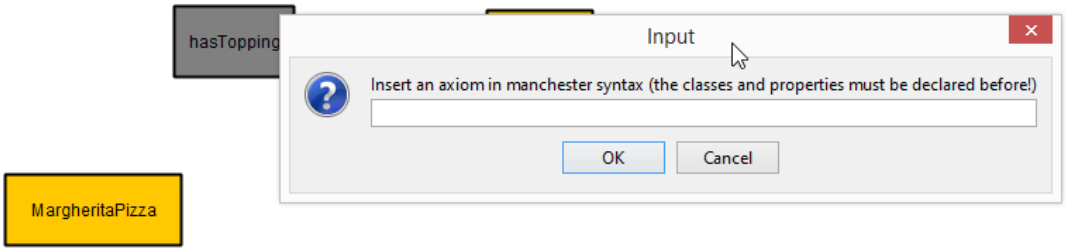


Figura 32 – Nova janela aparece quando selecionada a adição de axiomas em *Manchester Syntax*

Após confirmar a adição, o axioma é inserido na ontologia e mapeado para sua representação visual, observada na Figura 33.

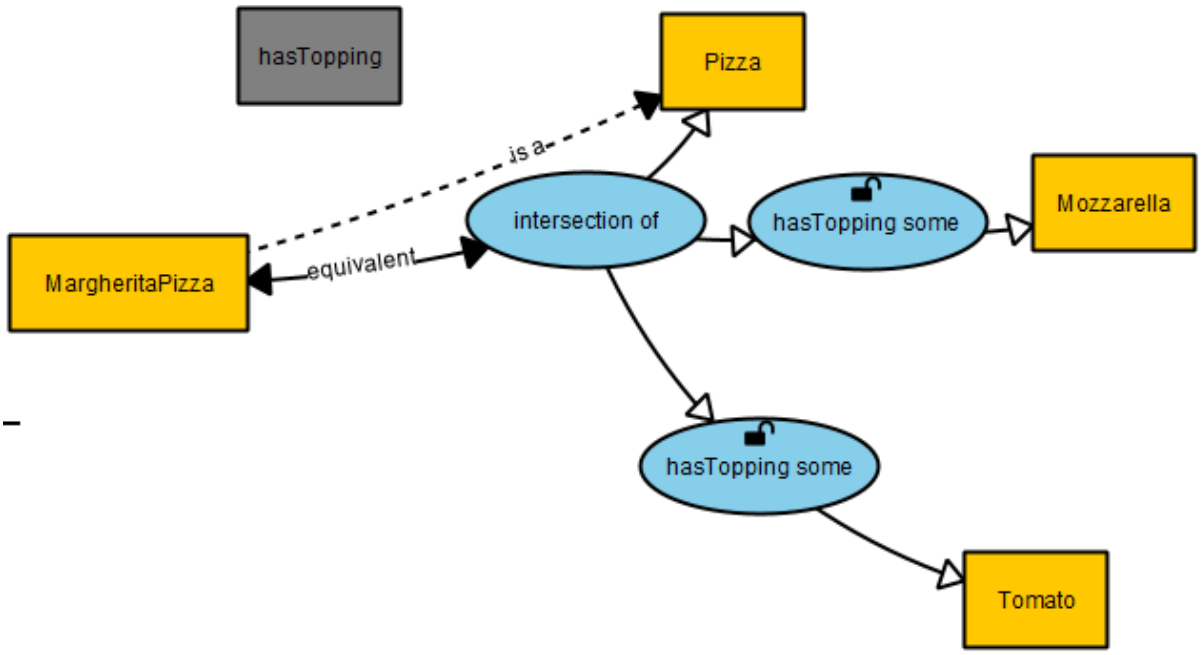


Figura 33 – Modelagem visual com adição automática de axioma em *Manchester Syntax*

Além da adição de axiomas em *Manchester Syntax*, o `Medina` também possui a funcionalidade de adição de *class expressions*. A adição ocorre de maneira análoga a adição de axioma: o usuário seleciona a opção *Add class expression* na paleta de ferramentas, em seguida adiciona na nova janela a *class expression* desejada e confirma a operação.

Considerando o axioma *MargheritaPizza EquivalentTo: Pizza and hasTopping some Mozzarella and hasTopping some Tomato*, poderia ser construído primeiro a *class expression Pizza and hasTopping some Mozzarella and hasTopping some Tomato* e em seguida criado o arco de axioma de equivalência entre MargheritaPizza e a interseção construída na *class expression* (Vide Figura 34).

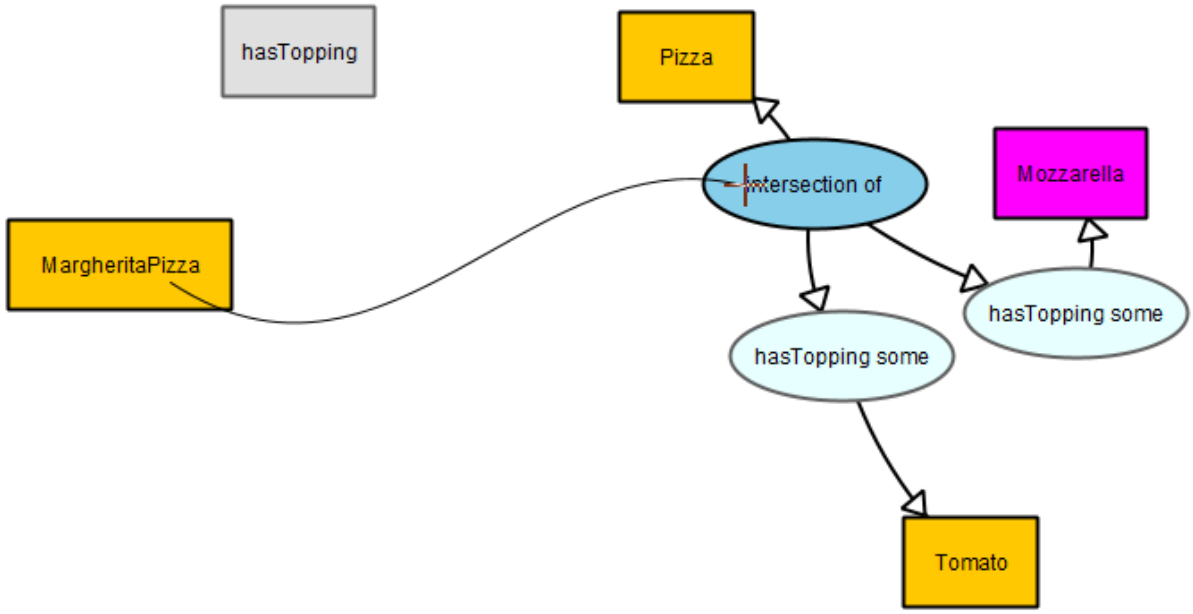


Figura 34 – Momento da adição de um axioma manual a uma *class expression* gerada a partir de sua representação em *Manchester Syntax*

3.3 ALGORITMOS

Nesta seção são apresentados os principais algoritmos que criamos e implementamos para o `Medina`. O primeiro algoritmo descrito é o *classExpression*, responsável por mapear nós do grafo representado visualmente em código OWL DL 2. O mapeamento é feito levando em consideração os nós dos tipos de classe e construtores de classe, tais como interseção, união, etc. através de uma busca em grafo, partindo do nó passado como entrada do algoritmo.

Com o algoritmo *classExpression* é possível representar classes complexas que podem ser utilizadas na construção de axiomas com expressividade alta. A construção de axiomas é realizada pelo segundo algoritmo a ser descrito: *addEdge*. O segundo algoritmo constrói a partir dos arcos do grafo axiomas tais como axiomas de subclasse ou equivalência. Com a construção de axiomas realizada pelo algoritmo *addEdge* o `Medina` verifica automaticamente, através de raciocínio, se existe alguma inferência ou inconsistência na ontologia. A verificação de possíveis deduções é feita pelo algoritmo *reasoning*, capaz de obter os raciocínios feitos pelo motor de raciocínio ativo e transferir de maneira apropriada para a representação em grafo do `Medina`.

3.3.1 Gerando *Class Expressions* através da modelagem visual

Uma das principais funcionalidades do `Medina` é a construção de código OWL DL 2 a partir da modelagem visual. Para que a modelagem visual possa ter uma maior flexibilidade, deve permitir a construção de classes complexas a partir da combinação dos construtores,

como por exemplo, representar $\ldots \sqcup \exists r.(A \sqcap B)$, o qual possui três construtores de classe aninhados (união, restrição existencial e interseção).

Para realizar o mapeamento da modelagem visual para código OWL DL 2, foi desenvolvido o Algoritmo 3.3.1, apresentado a seguir.

As entradas para o algoritmo são o grafo visual que está sendo modelado a partir da interface do `Medina` e o vértice pertencente ao grafo visual que se deseja mapear para OWL DL 2. A sua saída é a *class expression* equivalente ao vértice de entrada. De forma recursiva o algoritmo cria a *class expression* a partir das informações contidas no vértice e no grafo. Construir *class expressions* visualmente é possível utilizando os construtores de classes disponíveis como vértices do grafo e com o uso de arcos de **conexão**.

No início da execução, o algoritmo armazena todos os arcos do tipo conexão que saem do vértice a ser mapeado (N no Algoritmo 3.3.1). Logo depois, o algoritmo identifica qual o tipo do vértice a ser mapeado. Vértices do tipo união e interseção podem ter diversos arcos de conexão saindo de si, por isso é necessária a iteração sobre todos os arcos. Para cada arco é criada uma nova *class expression*, de forma recursiva, e adicionada a um conjunto de *class expressions*. Após a iteração sobre todos os arcos é então criado o código de união ou interseção, inserindo todas as *class expressions* retornadas.

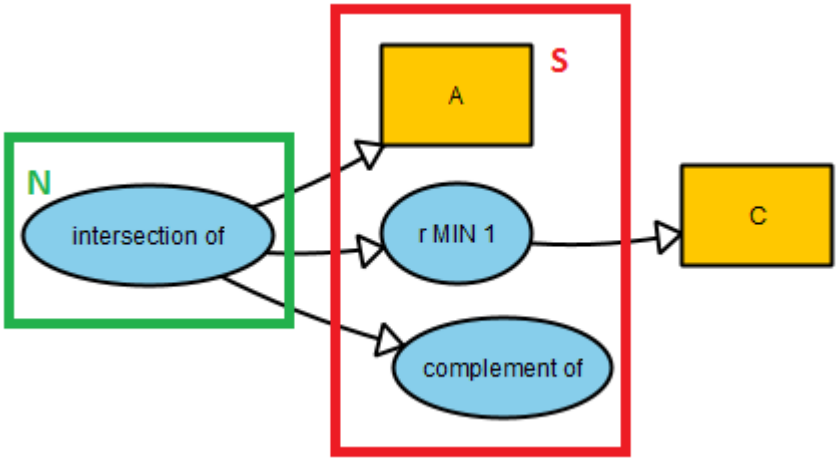


Figura 35 – Exemplo da implementação do algoritmo *class expression*

A Figura 35 representa visualmente a execução do algoritmo de geração da *class expression* $A \sqcap \geq 1\ r.\ C \sqcap \neg \top$. Na execução do algoritmo o valor de N é o construtor de interseção (raiz da árvore representada pela *class expression*) e o conjunto S, formado por todos os nós conectados ao construtor por arcos do tipo conexão e destacados na Figura 35. Para cada nó do conjunto S será executada a função *classExpression* recursivamente, atribuindo N para cada um dos elementos pertencentes a S.

Os vértices de entrada do Algoritmo 3.3.1 também podem ser vértices construtores de restrição sobre uma propriedade tais como restrição existencial ($\exists$), universal ($\forall$) e cardinalidades máxima ($\leq$), mínima ($\geq$) e igual ($=$). Para esses vértices construtores é

necessário armazenar qual a propriedade que a restrição está se referindo (armazenado em r no Algoritmo 3.3.1) e a cardinalidade (armazenada em n), se necessário.

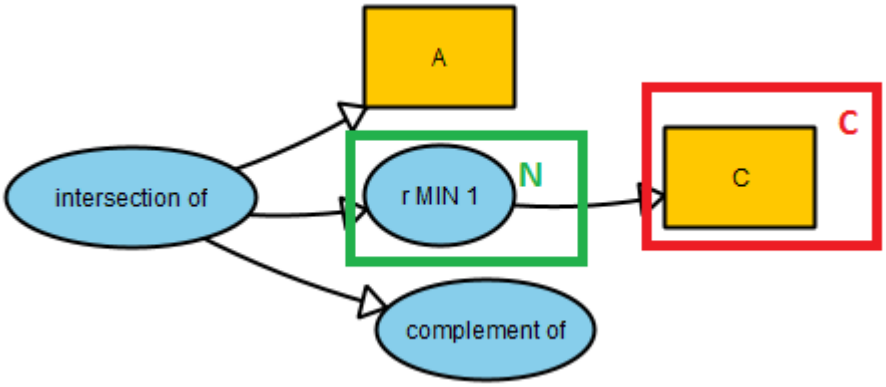


Figura 36 – Exemplo da implementação do algoritmo *class expression*.

Os vértices construtores de restrição, diferentemente dos de união e interseção, só podem ter um arco do tipo conexão saindo de si. O vértice de destino do arco, denominado C no algoritmo, é passado como parâmetro para a chamada do algoritmo *classExpression*. Para os vértices de cardinalidade, além das informações adquiridas nas restrições universal e existencial, ele também possui informação a respeito da cardinalidade (um valor numérico positivo). Na Figura 36 é representada a *class expression* $A \sqcap \geq 1\ r.\ C \sqcap \neg\ \top$, com foco na execução do algoritmo para o construtor de cardinalidade. Quando o algoritmo *classExpression* em execução estiver com N atribuído ao vértice de cardinalidade, serão armazenados os valores referentes a propriedade utilizada (r) e a cardinalidade (1) para construção do código da *class expression*. Através de recursividade, será obtida a *class expression* do próximo nó, conectado a N (por um arco de conexão) e denominado C.

Outro vértice mapeado para código é o de complemento. Semelhante aos vértices de restrição, o vértice de complemento deve possuir no máximo um arco de conexão saindo de si. Quando o algoritmo considerar N como o nó de complemento ($\neg$), será executado de maneira recursiva a verificação da *class expression* para o próximo nó conectado a ele, que neste caso será nulo dado que o nó de complemento não possui arcos de conexão a partir dele, conforme pode ser visto na Figura 37.

Caso o vértice seja nulo é então retornado por padrão a classe *owl:Thing*[8]. O vértice poderá ser nulo caso algum construtor não possua nenhum arco de conexão (caso do construtor de complemento representado na Figura 37). Portanto a partir do nó de complemento é gerada a *class expression* $\neg\ \top$.

Caso o vértice seja do tipo classe então o Algoritmo 3.3.1 retorna seu nome. Os vértices A e C, observados na Figura 37, são exemplos de vértices do tipo classe. Tratando-se do mapeamento entre a modelagem visual e o código OWL DL 2, os vértices de classe são as classes nomeadas, por isso a única informação necessária é o nome do vértice. Quando o

[8] Representada em DL como $\top$

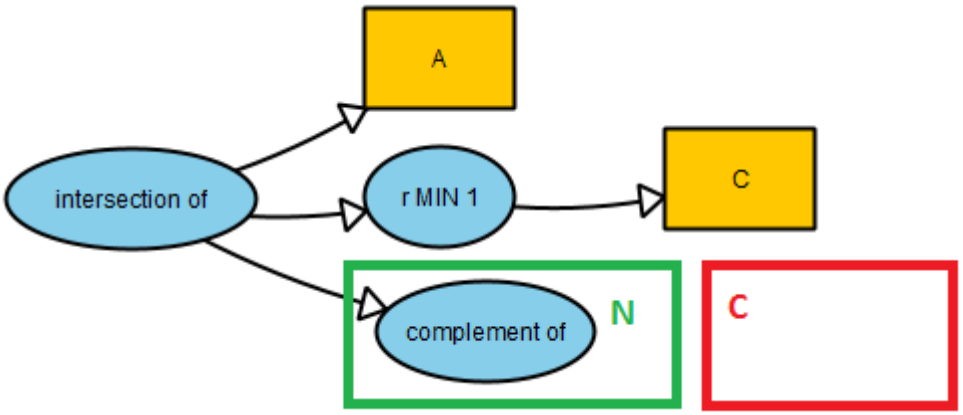


Figura 37 – Exemplo da implementação do algoritmo *class expression*.

vértice for nulo ou for do tipo classe a recursividade não é realizada e ambos são os casos-base do algoritmo. A seguir é apresentado o algoritmo *classExpression* em pseudocódigo.

Algoritmo 1: Algoritmo que converte *nodes* em *class expressions*

Require: Visual Graph G, node to get expression N

Ensure: Expression for N

```
1:  function classExpression(G,N)
2:      E ← G.getOutConnectionEdges(N)
3:      if N is an union or intersection node then
4:          create a empty set S
5:          for all e ∈ E do
6:              exp ← classExpression(G, e.destinationNode)
7:              add exp to S
8:          end for
9:          if N is an union node then
10:             return ObjectUnionOf(C_1 C_2 ... C_n), C_i ∈ S
11:         else
12:             return ObjectIntersectionOf(C_1 C_2 ... C_n), C_i ∈ S
13:         end if
14:     else if N is a restriction node {∃, ∀, ≥, ≤, =} then
15:         r ← N.property
16:         C ← classExpression(G, e.destinationNode), e ∈ E
17:         if N is a cardinality node {≥, ≤, =} then
18:             n ← N.cardinality
19:             if N is a max cardinality node then
20:                 return ObjectMaxCardinality(n r C)
21:             else if N is a min cardinality node then
22:                 return ObjectMinCardinality(n r C)
```

```
23:             else if N is an exact cardinality node then
24:                 return ObjectExactCardinality(n r C)
25:             end if
26:         else
27:             if N is an universal restriction node then
28:                 return ObjectAllValuesFrom(r C)
29:             else
30:                 return ObjectSomeValuesFrom(r C)
31:             end if
32:         end if
33:     else if N is a complement node then
34:         C ← classExpression(G, e.destinationNode), e ∈ E
35:         return ObjectComplementOf(C)
36:     else if N is a class node then
37:         return N.name
38:     else
39:         return owl : Thing
40:     end if
41: end function
```

Na Figura 38 é ilustrada a construção da *class expression* representada visualmente, em código OWL DL 2. Questões relacionadas a ciclos no grafo são evitados com a validação realizada no Algoritmo 3.3.2.

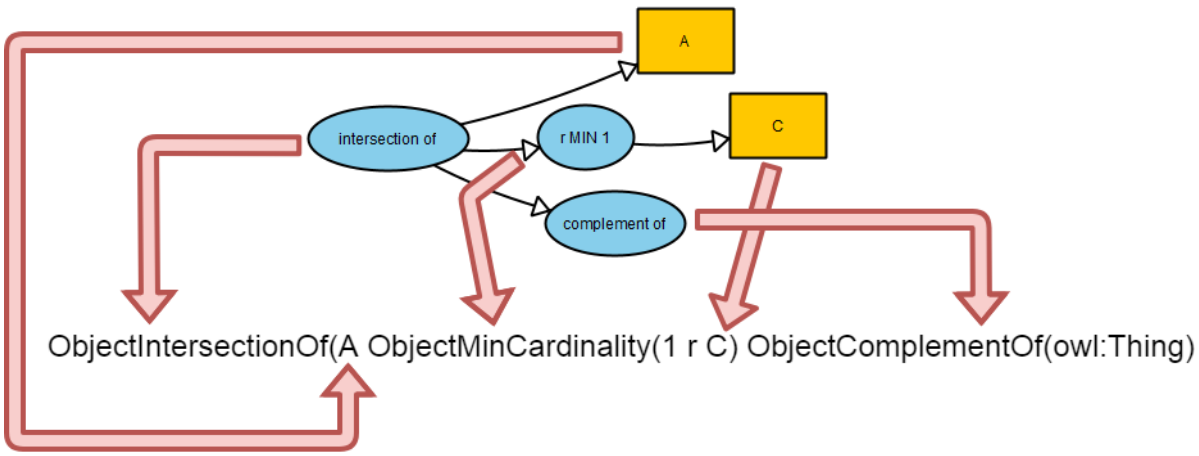


Figura 38 – Representação ilustrativa do mapeamento de uma *class expression* no `Medina`.

3.3.2 Adicionando axiomas na ontologia a partir da modelagem visual

A criação dos axiomas na ontologia ocorre em tempo de execução. Cada vez que um arco do tipo axioma é adicionado ou removido no grafo visual é realizada uma operação equivalente na ontologia. O Algoritmo 3.3.2 foi desenvolvido para adicionar o arco visualmente

no grafo e ao mesmo tempo adicionar seu equivalente na ontologia, garantindo assim a sincronia entre ambos.

A princípio o algoritmo realiza a computação somente com arcos válidos. Conforme visto anteriormente, os vértices e arcos do grafo visual possuem restrições quanto a seu uso, alguns arcos devem sair de vértices específicos bem como devem também chegar a vértices específicos de acordo com o seu tipo. Na Tabela 3 é possível ver todas as restrições para a criação de arcos visualmente.

Tabela 3 – Tabela de restrições dos arcos.

Arco	**Vértices de saída**	**Vértices de entrada**
SubClasse	Classes, conectores e restrições	Classes, conectores e restrições
Equivalência	Classes, conectores e restrições	Classes, conectores e restrições
Disjunção	Classes, conectores e restrições	Classes, conectores e restrições
SubPropriedade	Propiedades	Propriedades
Propriedade Inversa	Propriedades	Propriedades
Domínio	Propriedades	Classes, conectores e restrições
Imagem	Propriedades	Classes, conectores e restrições
Asserção de Classes	Indivíduos	Classes, conectores e restrições
Asserção de Propriedades	Indivíduos	Indivíduos
Conexão	Conectores e restrições	Classes, conectores e restrições

Um arco é considerado válido se ele atende as restrições quanto aos vértices. Caso o arco seja válido o algoritmo separa os arcos inseridos manualmente e os inferidos pelo motor de raciocínio. Arcos inferidos são adicionados ao grafo visual para serem apresentados ao usuário, porém não são adicionados na ontologia pois já foram deduzidos dela.

Os axiomas, na sintaxe da OWL DL 2 e na ferramenta, podem ser separados em tipos: axiomas com *class expressions*, com propriedades e de asserção. Na primeira parte do Algoritmo 3.3.2 são mapeados os axiomas com *class expressions*. Esses axiomas, utilizam *class expressions* em sua construção. No `Medina` estão disponíveis os axiomas de equivalência, subclasse e de disjunção. Os arcos de equivalência, subclasse e de disjunção, conforme a Tabela 3, possuem como vértices de entrada e saída construtores de classe. Conforme visto no Algoritmo 3.3.1, os vértices de construtores e de classe são convertidos em *class expressions*.

Os axiomas com propriedades utilizados na ferramenta são de domínio, imagem, subpropriedade e propriedade inversa. Os arcos desses axiomas possuem como saída vértices do tipo propriedade e para a construção do axioma é obtido o nome do vértice para ser mapeado em uma propriedade OWL DL 2 de mesmo nome. Os arcos de domínio e imagem possuem como chegada construtores ou classes e podem ser convertidos em *class expressions*. Já os arcos de subpropriedade e propriedade inversa possuem como chegada vértices de propriedade onde também é utilizado o nome para a construção do axioma.

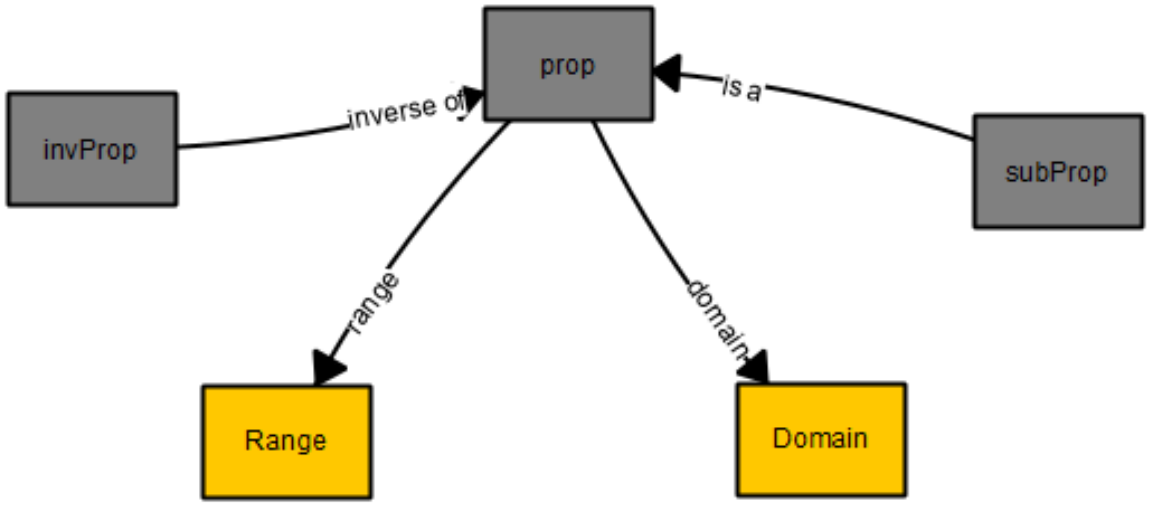


Figura 39 – Exemplo contendo os axiomas relacionados a propriedades no `Medina`

Na Figura 39 tem como exemplo os axiomas de domínio, imagem, subpropriedade e propriedade inversa representados como arcos com os nomes *domain*, *range*, *is a* e *inverse of* respectivamente. Na iteração do algoritmo *addEdge*, tomando como exemplo o arco de propriedade inversa (nomeado *inverse of* na Figura 39), são obtidos os nomes dos vértices de origem e destino do arco, *invProp* e *prop* respectivamente. Após adquirir os nomes dos nós do tipo propriedade é então construído o axioma em OWL DL 2, no algoritmo representado em sintaxe funcional[9]: *InverseObjectProperties(invProp prop)*. Na Figura 40 é possível observar de maneira ilustrativa o mapeamento dos elementos visuais na construção do código da ontologia.

Axiomas de asserção, na ferramenta proposta, são os arcos que possuem como saída vértices do tipo indivíduo e semelhante aos vértices de propriedade é utilizado o seu nome para a construção do axioma. Arcos de asserção de classe possuem como entrada vértices construtores e de classe, que para serem construídos axiomas os utilizando, são convertidos para *class expressions*. Para a construção dos axiomas de propriedades, além de utilizar o nome do vértice de saída são utilizados o nome de vértice de entrada e o nome do arco, que representa o nome da propriedade a ser feita a asserção.

[9] Representação alternativa de ontologias OWL

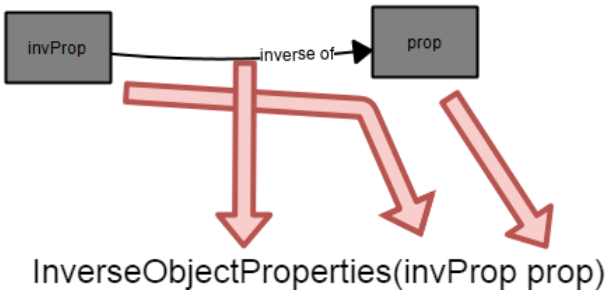


Figura 40 – Representação ilustrativa do mapeamento de um axioma no `Medina`.

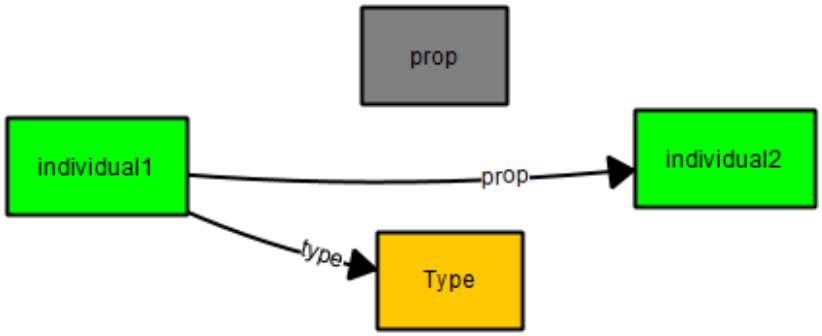


Figura 41 – Exemplo contendo os axiomas relacionados a indivíduos no `Medina`

Na Figura 41 existem exemplos dos axiomas de asserção de classe e de propriedade. O axioma de classe está representado como o arco nomeado *type*, ligando os vértices *individual1* e *Type*. Já o axioma de asserção de propriedade é o arco com nome *prop* (nome da propriedade declarada) ligando os indivíduos *individual1* e *individual2*.

Caso os arcos a serem adicionados sejam referentes a axiomas, o algoritmo realiza o mapeamento para axiomas em OWL DL 2 conforme descrito acima. Já para arcos de conexão, desenvolvidos especialmente para permitir a construção de classes complexas, a execução é diferente. É realizada uma busca sobre os arcos e vértices conectados ao arco em questão com finalidade de detectar se ele faz parte de algum axioma já mapeado. Caso esteja conectado a algum arco de axioma, o axioma é removido da ontologia e do grafo visual, para que seja adicionado o arco de conexão e em seguida adicione novamente o axioma e seu arco na ontologia e no grafo visual respectivamente. Essa abordagem é utilizada para garantir que o grafo visual e ontologia estejam sincronizados. A seguir é apresentado o algoritmo *addEdge* em pseudocódigo.

Algoritmo 2: Algoritmo que adiciona axiomas OWL a partir de arcos do grafo visual

Require: Visual Graph G, Ontology O and the edge to add e.

```
1: procedure ADDEDGE(G,O,e)
2:     if e is a valid edge then
3:         if e is not an inferred edge then
4:             initialize ax
5:             if e is a class expression axiom edge {≡, ⊑, ⊑ ¬} then
```

6: $C_1 \leftarrow classExpression(edge.sourceNode)$
7: $C_2 \leftarrow classExpression(edge.destinationNode)$
8: **if** e is an equivalent axiom edge **then**
9: $ax \leftarrow EquivalentClasses(C_1\ C_2)$
10: **else if** e is a subclass axiom edge **then**
11: $ax \leftarrow SubClassOf(C_1\ C_2)$
12: **else if** e is a disjoint axiom edge **then**
13: $ax \leftarrow DisjointClasses(C_1\ C_2)$
14: **end if**
15: **else if** e is a property axiom edge **then**
16: $R_1 \leftarrow e.sourceNode.name$
17: **if** e is a domain or range axiom edge **then**
18: $C \leftarrow classExpression(e.destinationNode)$
19: **if** e is a domain axiom edge **then**
20: $ax \leftarrow ObjectPropertyDomain(R_1\ C)$
21: **else if** e is a range axiom edge **then**
22: $ax \leftarrow ObjectPropertyRange(R_1\ C)$
23: **end if**
24: **else if** e is a subproperty or inverse property axiom edge **then**
25: $R_2 \leftarrow e.destinationNode.name$
26: **if** e is a subproperty axiom edge **then**
27: $ax \leftarrow SubObjectPropertyOf(R_1\ R_2)$
28: **else if** e is an inverse property axiom edge **then**
29: $ax \leftarrow InverseObjectProperties(R_1\ R_2)$
30: **end if**
31: **else if** e is an assertion edge **then**
32: $i_1 \leftarrow e.sourceNode.name$
33: **if** e is a class assertion edge **then**
34: $C \leftarrow classExpression(e.destinationNode)$
35: $ax \leftarrow ClassAssertion(C\ i_1)$
36: **else**
37: $i_2 \leftarrow e.destinationNode.name$
38: $R \leftarrow e.name$
39: $ax \leftarrow ObjectPropertyAssertion(R\ i_1\ i_2)$
40: **end if**
41: **else**
42: $N \leftarrow e.sourceNode$
43: $axEdge \leftarrow e$
44: **while** $axEdge$ is a connection node or axEdge is null **do**

45: $axEdge \leftarrow G.getInEdge(N)$
46: $N \leftarrow axEdge.sourceNode$
47: **end while**
48: **if** $axEdge$ is not null **then**
49: removeEdge(G,O,axEdge)
50: $G \leftarrow G \cup \{e\}$
51: addEdge(G,O,axEdge)
52: **else**
53: $G \leftarrow G \cup \{e\}$
54: **end if**
55: end execution here
56: **end if**
57: **end if**
58: $O \leftarrow O \cup \{ax\}$
59: **end if**
60: $G \leftarrow G \cup \{e\}$
61: **end if**
62: **end procedure**

3.3.3 Apresentando o raciocínio automático visualmente

Após cada atualização visual no grafo, a ferramenta realiza novos raciocínios, de maneira automática. Internamente, a ferramenta realiza mapeamentos entre as entidades do grafo e as entidades da ontologia. Axiomas são representados como arcos enquanto classes, propriedades, indivíduos e os construtores de classes são representados como vértices.

O Algoritmo 3.3.3 possui como entrada o grafo visual e a ontologia em execução. Inicialmente, uma nova instância do motor de raciocínio (`Pellet` por exemplo) é criada e são armazenadas as classes insatisfatíveis a partir da execução do motor de raciocínio.

Para cada uma das classes é mapeado o vértice equivalente, através do nome que ambos possuem, e é atribuído a esse vértice que sua classe mapeada é insatisfatível. Ao indicar a insatisfatibilidade, a ferramenta altera a cor do vértice para vermelho, como pode ser visto na Figura 42. Em seguida são armazenados os axiomas que explicam a insatisfatibilidade da classe em questão utilizando o motor de raciocínio instanciado anteriormente.

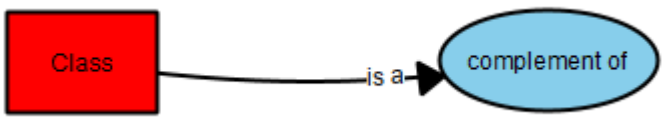


Figura 42 – Exemplo contendo uma insatisfatibilidade

Para cada um dos axiomas obtidos o algoritmo mapeia o arco de axioma equivalente

no grafo e o referencia com o vértice da classe insatisfatível. Dessa maneira, quando o usuário passar o *mouse* sobre uma classe insatisfatível será possível também apresentar ao usuário os axiomas que causam a insatisfatibilidade (Vide Seção 3.2.5).

Além de apresentar visualmente a insatisfatibilidade das classes, o Algoritmo 3.3.3 também é responsável por mapear as inferências em OWL DL 2 para novos arcos presentes visualmente no grafo. Os axiomas de inferência disponíveis são os de subclasse, domínio, imagem, asserção de classes e subpropriedade. Para cada um dos axiomas é criado um novo arco do tipo inferência – representado visualmente pontilhado.

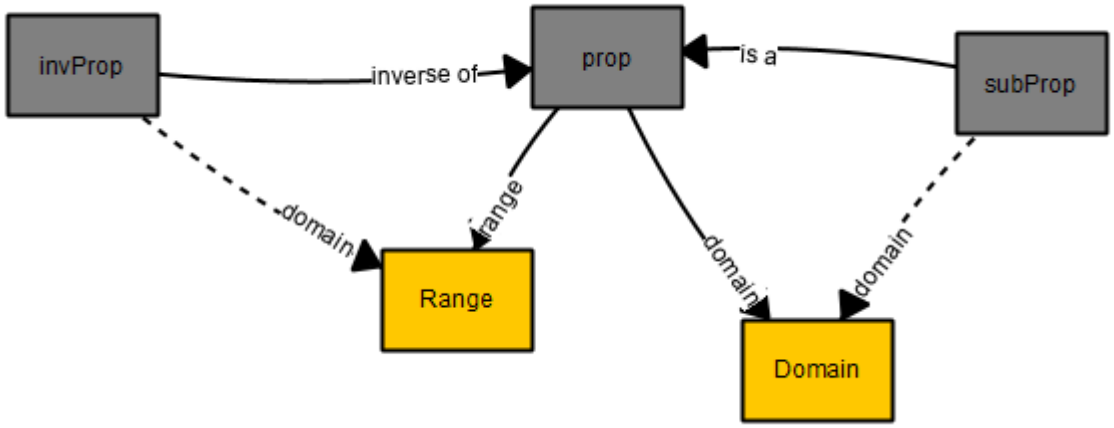


Figura 43 – Exemplo contendo duas inferências de domínio.

Na Figura 43 pode ser observada duas inferências do tipo domínio. Todos os relacionamentos com a propriedade *prop* devem partir de indivíduos que sejam do tipo *Domain*, consequentemente se *subProp* é uma subpropriedade de *prop*, então todos os indivíduos de *subProp* também devem ter relacionamentos partindo exclusivamente de indivíduos do tipo *Domain*. Se todos os indivíduos que possuem uma relação de *prop* devem se conectar com indivíduos do tipo *Range* e *prop* e *invProp* são propriedades inversas, então todos os indivíduos que possuem relações partindo *invProp* devem ser do tipo *Range*. Por isso foram inferidos os dois axiomas de domínio, representados com arcos pontilhados na Figura 43.

Por fim, caso seja criado algum arco de inferência, este é adicionado ao grafo visualmente e, conforme visto no Algoritmo 3.3.2, ele tem um tratamento diferenciado dos outros arcos, não sendo adicionado à ontologia. A seguir é apresentado o algoritmo *reasoning* em pseudocódigo.

Algoritmo 3: Algoritmo que representa os axiomas deduzidos de maneira visual
Require: Visual Graph G and Ontology O

1: **procedure** REASONING(G,O)
2: create a new reasoner instance for ontology O and put into R
3: $U \leftarrow R.unsatisfiableClasses$

4: **for all** $u \in U$ **do**
5: $n \leftarrow G.findNode(u)$
6: $n.satisfiability \leftarrow$ **False**
7: get axioms that explain the unsatifiability of u and put into A
8: create an empty edge set and put into E
9: **for all** $ax \in A$ **do**
10: $e \leftarrow G.edgeForAxiom(ax)$
11: add e to E
12: **end for**
13: $n.explanationEdges \leftarrow E$
14: **end for**
15: $I \leftarrow R.inferredAxioms$
16: **for all** $i \in I$ **do**
17: create an edge inferred axiom and put into e
18: **if** i is a subclass axiom **then**
19: $e.type \leftarrow$ subclass
20: $e.sourceNode \leftarrow G.findNode(i.subClass)$
21: $e.destinationNode \leftarrow G.findNode(i.subClass)$
22: **else if** i is a domain axiom **then**
23: $e.type \leftarrow$ domain
24: $e.sourceNode \leftarrow G.findNode(i.property)$
25: $e.destinationNode \leftarrow G.findNode(i.class)$
26: **else if** i is a range axiom **then**
27: $e.type \leftarrow$ range
28: $e.sourceNode \leftarrow G.findNode(i.property)$
29: $e.destinationNode \leftarrow G.findNode(i.class)$
30: **else if** i is a subproperty axiom **then**
31: $e.type \leftarrow$ subproperty
32: $e.sourceNode \leftarrow G.findNode(i.subProperty)$
33: $e.destinationNode \leftarrow G.findNode(i.superProperty)$
34: **else if** i is an inverse property axiom **then**
35: $e.type \leftarrow$ inverse property
36: $e.sourceNode \leftarrow G.findNode(i.property)$
37: $e.destinationNode \leftarrow G.findNode(i.inverseProperty)$
38: **else if** i is a class assertion axiom **then**
39: $e.type \leftarrow$ class assertion
40: $e.sourceNode \leftarrow G.findNode(i.individual)$
41: $e.destinationNode \leftarrow G.findNode(i.class)$
42: **end if**

43: get axioms that explain i and put into A
44: create an empty edge set and put into E
45: **for all** $ax \in A$ **do**
46: $e \leftarrow G.edgeForAxiom(ax)$
47: add e to E
48: **end for**
49: $e.explanationEdges \leftarrow E$
50: $addEdge(G, O, e)$
51: **end for**
52: **end procedure**

3.4 CONSIDERAÇÕES FINAIS

Neste capítulo foi apresentado de maneira geral o funcionamento do `Medina`, uma ferramenta para modelagem visual de ontologias com uso de raciocínio automático que constrói ontologias OWL DL 2 com expressividade máxima $\mathcal{ALCHIQ}$.

Foram descritos os componentes do sistema proposto e as tecnologias utilizadas. Além disso, foi também apresentado exemplos de funcionamento do sistema utilizando suas principais funcionalidades.

O `Medina` foi desenvolvido para dar suporte a engenheiros de ontologias e também especialistas de domínio, provendo funcionalidades de raciocínio automático e exposição de axiomas que causam os raciocínios obtidos como forma de *feedback* em tempo real na construção das ontologias. Os principais benefícios e diferenciais da arquitetura do `Medina` são:

- Com a arquitetura do sistema proposto, foi possível prover a visualização e edição de ontologias simultaneamente, gerando automaticamente código OWL DL 2 em tempo real e verificando os fatos presentes na ontologia através de processos de raciocínio.

- Os componentes foram desenvolvidos levando em consideração o reuso. Dessa forma, os componentes podem ser utilizados em outras soluções na construção de ontologias (*Ontology Core*), na representação de conhecimento de maneira visual (*Medina Graph*) ou na utilização de outros motores de raciocínio (*Reasoner*).

- A representação visual das ontologias no `Medina` foi desenvolvida levando em consideração a representação visual de inferências (arcos pontilhados) e classes insatisfatíveis (classes em vermelho), facilitando a compreensão sobre os raciocínios obtidos no desenvolvimento das ontologias, diferentemente da maioria dos sistemas de visualização de ontologias.

- A combinação de representações visuais em grafo e em árvore (semelhante ao `Protégé`) proporciona aos usuários o melhor das duas técnicas. A capacidade de ocultar axiomas, mas ainda mantê-los na ontologia representados dentro da classe que o utiliza, caracteriza um benefício para os usuários desenvolverem ontologias grandes comprometendo a visualização da ontologia de forma mínima.
- Uma das principais funcionalidades da ferramenta proposta é a *visual explanation* de inferências. O *Medina* é o único sistema (Vide Capítulo 5) a possuir tal funcionalidade, sendo capaz de beneficiar o desenvolvimento de ontologias por usuários de diferentes níveis de experiência em construção de ontologias e entendimento de lógica.

No capítulo a seguir, apresentamos a questão de pesquisa, as hipóteses, os experimentos realizados em relação ao `Medina` e os resultados obtidos que validam as hipóteses.

4 EXPERIMENTOS

4.1 QUESTÕES DE PESQUISA E HIPÓTESES

O desenvolvimento de ontologias, utilizando a linguagem OWL 2 é considerada por algumas pessoas de difícil aprendizagem, tanto em relação a sua sintaxe quanto a sua semântica. Associado a essas dificuldades a curva de aprendizado, o custo de tempo e consequentemente de recursos financeiros atenuam a problemática em torno da engenharia de ontologias.

Visto o problema relacionado ao desenvolvimento de ontologias, foi definida a seguinte questão de pesquisa:

De que maneira uma ferramenta de autoria inteligente para modelagem de ontologias de forma visual e que realiza raciocínio automático pode ser útil para o aprendizado, construção e manutenção de bases de conhecimento modeladas como ontologias?

Para tentar atender devidamente a esta questão, temos uma hipótese que se reporta ao objetivo inicial apresentado no capítulo de introdução (Capítulo 1) desta dissertação.

Desta forma, a hipótese definida foi:

- **Hipótese de Pesquisa** – H_1: "`Medina` provê **capacidade**, **corretude** e **facilidade** na construção de ontologias expressivas e provê **capacidade** de realizar raciocínio de subsunção e checagem de inconsistências a partir das interações em modelagem visual".

4.2 EXPERIMENTAÇÃO

Para verificar a validade da hipótese foram conduzidos dois experimentos. A seguir serão apresentados o método para a construção dos experimentos, ameaças à validação, os resultados obtidos e uma discussão sobre os mesmos.

4.2.1 Método

Os experimentos realizados neste trabalho foram divididos em duas etapas, Experimento 1 e Experimento 2. O Experimento 1, por sua vez, foi dividido em três partes. Na primeira e segunda parte do Experimento 1, nove alunos da disciplina de Inteligência Artificial do curso de Ciência da Computação da Universidade Federal Rural de Pernambuco, Unidade Acadêmica de Garanhuns (UFRPE-UAG) do semestre 2018.1, participaram de aulas relacionadas a ontologias e raciocínio automático. Após as aulas foram aplicados questionários com objetivo de verificar a compreensão dos alunos quanto ao aprendizado e percepção de possíveis raciocínios.

Seis dos nove alunos das partes 1 e 2 participaram da parte 3 do Experimento 1, divididos em grupos para utilizarem as ferramentas `Medina` e `Protégé` (MUSEN, 2015). Ambos os grupos receberam uma ontologia modelada em Lógica de Descrições, para ser modelada nas duas ferramentas. Na construção da ontologia nas duas ferramentas, através da observação da execução nesta parte do experimento pelos participantes, foram obtidas informações referentes a participação dos integrantes do grupo na modelagem, possíveis boas práticas realizadas pelos usuários, a interação dos alunos com os sistemas utilizados e o tempo gasto na modelagem.

O Experimento 2 foi realizado com estudantes da disciplina Tópicos Avançados em Inteligência Artificial do curso de Ciência da Computação da Universidade Federal Rural de Pernambuco, Unidade Acadêmica de Garanhuns (UFRPE-UAG) do semestre 2018.2, utilizando o `Medina`, e também o `Protégé` para construção de alguns axiomas em DL. As interações dos estudantes com as ferramentas foram registradas através de *softwares* de captura de tela, obtendo informações sobre a forma como os estudantes utilizaram as ferramentas e possibilitando a obtenção de dados referentes ao tempo gasto na modelagem e o percentual de axiomas corretos. Por fim foram aplicados dois questionários para os estudantes com perguntas relacionadas ao desenvolvimento da ontologia em cada ferramenta e com perguntas relacionadas especificamente a ferramenta proposta. Os objetivos deste experimento foi verificar a capacidade, corretude e facilidade na construção de ontologias expressivas a partir de modelagem visual e verificar a capacidade do sistema proposto em realizar raciocínios de inferência e inconsistências automaticamente.

Os estudantes participantes dos experimentos, após as aulas aplicadas na parte 1 do Experimento 1, obtiveram conhecimento de Lógica de Descrições, sendo capazes de entender e desenvolver axiomas com os construtores de expressividade $\mathcal{ALC}$. Até a realização dos experimentos, os alunos tiveram contato com o `Protégé` e `Medina` apenas em sala de aula, em exemplos apresentados pelo seu professor.

O `Protégé` foi utilizado nos experimentos como uma forma de relacionar os resultados da ferramenta proposta com a ferramenta referência no desenvolvimento de ontologias. Dessa forma é possível verificar não somente a capacidade, corretude e facilidade do `Medina`, mas também verificar o quanto a capacidade, corretude e facilidade do sistema proposto é melhor, ou pior, que o *software* mais utilizado para a construção e manutenção de ontologias.

Com a realização dos experimentos é esperado que dificuldades relacionadas a semântica e sintaxe do OWL DL 2 e entendimento dos raciocínios obtidos sejam reduzidos com o uso da ferramenta proposta em relação ao `Protégé` e que as funcionalidades como visualização de inferências e insatisfatibilidade sejam relevantes para a modelagem dos usuários.

4.2.2 Ontologias Utilizadas nos Experimentos

No Experimento 1 foram utilizados axiomas de uma ontologia que expressassem diversos níveis de dificuldade na sua construção, sem necessariamente possuir engajamento ontológico. O objetivo deste experimento foi verificar como e em quanto tempo os grupos desenvolviam a ontologia usando as duas ferramentas indicadas na Seção anterior (`Medina` e `Protégé`). Os seguintes axiomas foram utilizados na construção da ontologia O_0 e apresentados aos participantes do Experimento 1:

$$Bat \equiv Mamifero \sqcap \forall possui.Asas \tag{4.1}$$
$$Mamifero \sqsubseteq Animal \sqcap \neg\exists poe.Ovos \sqcap \forall possui.Mamas \tag{4.2}$$
$$Passaro \equiv Vertebrado \sqcap \forall possui.Ossos \sqcap \forall possui.Asas \sqcap \exists possui.Pena \tag{4.3}$$
$$Humano \equiv Vertebrado \sqcap \neg\forall possui.Pena \tag{4.4}$$
$$Mamifero \sqsubseteq Vertebrado \sqcap \exists live.(AmbTerrestre \sqcup AmbAquatico) \tag{4.5}$$
$$Surf \sqsubseteq EspAquatico \tag{4.6}$$
$$Surf \sqsubseteq \neg Skate \tag{4.7}$$
$$Surf \sqsubseteq EspRadical \tag{4.8}$$
$$Skate \sqsubseteq EspRadical \tag{4.9}$$
$$Surfista \equiv \forall pratica.Yoga \tag{4.10}$$
$$Surfista \equiv \forall utiliza.Prancha \tag{4.11}$$
$$Surfista \equiv \forall treina.Musculacao \tag{4.12}$$
$$Surfista(Ryan) \tag{4.13}$$
$$Surfista(Elyson) \tag{4.14}$$
$$Humano \sqsubseteq \exists pratica.EspRadical \sqcap \exists treina.Musculacao \tag{4.15}$$

Para a realização do Experimento 2 foram extraídos axiomas da ontologia NTDO[1] (SANTANA et al., 2011), ontologia que modela o domínio das doenças tropicais negligenciadas, tais como dengue e leishmaniose. A ontologia foi dividida em diversos subconjuntos presentes em diferentes arquivos OWL. Foram extraídos os seguintes axiomas, representados em DL, do arquivo ntdo.owl (com exceção do Axioma 4.21) e utilizados para a construção da ontologia O_1 utilizada no experimento:

$$\begin{aligned} Process \sqsubseteq & (\exists hasDuration.TimeInterval) \\ & \sqcup (\exists hasPointInTime.PointInTime) \\ & \sqcup (= 1\ projectsTo.Chronoid) \end{aligned} \tag{4.16}$$

[1] <http://cin.ufpe.br/~ntdo/>

$$\begin{aligned} HealthSurveillanceNotificationAction \equiv\ & Action \\ & \sqcap (\exists precededBy.(PathologicalProcess \sqcup DeathEvent)) \\ & \sqcap (\exists hasGeographicLocation.SpatialRegion) \end{aligned} \tag{4.17}$$

$$LivingOrganism \sqsubseteq Organism \tag{4.18}$$

$$\begin{aligned} DeathEvent \sqsubseteq & InstantaneousProcess \\ & \sqcap \exists hasDeathPatient.LivingOrganism \\ & \sqcap \exists precededBy.BiologicalDeathProcess \\ & \sqcap \exists hasGeographicLocation.SpatialRegion \\ & \sqcap = 1\ hasPrimaryDeathCause.Process) \end{aligned} \tag{4.19}$$

$$\begin{aligned} InjuryEvent \sqsubseteq & InstantaneousProcess \\ & \sqcap \forall causedBy.(Process \sqcap \neg BiologicalProcessualEntity) \\ & \sqcap \forall hasInjuriedPatient.LivingOrganism \\ & \sqcap \forall hasGeographicLocation.SpatialRegion) \end{aligned} \tag{4.20}$$

$$DeathEvent \sqsubseteq \neg InstantaneousProcess \tag{4.21}$$

Para obter um melhor entendimento sobre como foram modeladas as ontologias pelos participantes, foi definida também uma segunda ontologia (O_2), contendo axiomas dos subconjuntos *Leishmaniasis.owl* e *DengueFever.owl* adicionado o axioma 4.25 ao experimento:

$$\begin{aligned} & LeishmaniaTransferByVector\_3 \equiv Transfer \\ & \sqcap \exists hasAgent.(LutzomyiaCarreraiCarrerai \\ & \quad \sqcup LutzomyiaFlaviscutellata \sqcup LutzomyiaLongipalpis) \\ & \sqcap \exists hasPatient.(\exists physicallyLocatedIn.Human \\ & \quad \sqcap (LeishmaniaAmazonensis \sqcup LeishmaniaBraziliensis \\ & \quad \quad \sqcup LeishmaniaChagasi \sqcup LeishmaniaLlanosmartini \\ & \quad \quad \sqcup LeishmaniaYucumensis)) \\ & \sqcap \exists hasGeographicLocation.BoliviaLocation \end{aligned} \tag{4.22}$$

$$\begin{aligned} DengueFever \equiv & PathologicalProcess \sqcap \exists causedBy.(\\ & \quad DEN1 \sqcup DEN2 \sqcup DEN3 \sqcup DEN4) \\ & \sqcap \exists hasLocus.Human \\ & \sqcap \exists realizationOf.DengueFeverDisposition \end{aligned} \tag{4.23}$$

$$\begin{aligned} DengueVirusTransferByVector \sqsubseteq \forall hasAgent.AedesAegypti \\ \sqcap \forall hasPatient.(\\ (DEN1 \sqcup DEN2 \sqcup DEN3 \sqcup DEN4) \\ \sqcap \forall causes.DengueFever) \\ \sqcap \forall hasGeographicLocation.SpatialRegion \end{aligned} \tag{4.24}$$

$$DengueFever \sqsubseteq \neg PathologicalProcess \tag{4.25}$$

Os Axiomas 4.21 e 4.25 foram acrescentados ao conjunto de axiomas a fim de forçar insatisfatibilidades na modelagem. Métricas relacionadas as ontologias definidas previamente nesta seção, tais como quantidade de classes, quantidade de propriedades, quantidade de indivíduos e quantidade axiomas lógicos são apresentadas na Tabela 5.

Tabela 5 – Métricas das ontologias utilizadas no experimento

Ontologia	O_0	O_1	O_2
Qtd. de Classes	21	15	22
Qtd. de Propriedades	7	9	8
Qtd. de Indivíduos	2	0	0
Qtd. de Axiomas	15	6	4

Além da construção dos axiomas das ontologias O_1 e O_2 os participantes do Experimento 2 realizaram tarefas relacionadas ao entendimento do que está sendo modelado. As tarefas foram:

- Verificar a existência de alguma insatisfatibilidade na ontologia e citar quais são, caso existam.
- Verificar quais axiomas implicam em cada uma das insatisfatibilidades.
- Verificar a existência de alguma inferência na ontologia e citar quais são, caso existam.
- Verificar quais axiomas implicam em cada uma das inferências existentes.

4.2.3 Ameaças à Validade

Quatro categorias de ameaças a validade são consideradas neste estudo (WOHLIN et al., 2012): validade interna, validade externa, confiabilidade e validade de construção.

- **Validade Interna:** A validade interna é focada na validação do estudo atual e provavelmente não é um problema. Todos os estudantes da disciplina de Inteligência

Artificial da UFRPE-UAG participaram do experimento, ou seja, todos os possíveis participantes do experimento fizeram parte do estudo. No entanto, ameaças a validade interna devem ser consideradas:

1. Os participantes do experimento podem ficar cansados ao longo da execução do experimento (provavelmente 30 minutos por modelagem) e influenciar de maneira negativa nos dados obtidos, especialmente na corretude e no tempo de modelagem.
2. Na parte 3 do primeiro experimento, os estudantes desenvolveram a mesma ontologia O_0 nas ferramentas `Medina` e `Protégé`, caracterizando uma possível ameaça à validade interna devido ao uso da mesma ontologia no desenvolvimento o que possibilita aprendizado sobre o que está sendo modelado por parte dos participantes. Esta ameaça é minimizada na execução do Experimento 2 pois são utilizadas duas ontologias distintas na execução (O_1 e O_2).
3. Como as diversas partes do experimento ocorrem em momentos distintos alguns estudantes podem não participar das partes seguintes. Três estudantes que participaram da primeira parte não participaram da execução da segunda parte. E na realização do Experimento 2 mais um aluno não participou, restando cinco estudantes para a execução do segundo experimento. Mesmo com a desistência de parte dos estudantes, a quantidade restante dos participantes foi significativa em relação ao total de alunos, participando do último experimento cinco alunos que correspondem a mais da metade (50%) do total de estudantes que participaram no início do estudo.

- **Validade Externa:** A validade externa está associada a capacidade de generalizar os resultados do experimento. Pode ser dividida em relação aos alunos da disciplina de Inteligência Artificial, do curso de Ciência da Computação da UFRPE-UAG posteriores a realização do experimento, estudantes de outras disciplinas de Inteligência Artificial em geral e desenvolvedores de ontologias em geral. É muito provável que resultados similares sejam obtidos ao executar este experimento com novos estudantes da disciplina de Inteligência Artificial, do curso de Ciência da Computação da UFRPE-UAG. Os resultados obtidos a partir das análises podem provavelmente ser generalizados para outras disciplinas de Inteligência Artificial uma vez que o *background* dos estudantes é semelhante. Entretanto, a generalização para desenvolvedores de ontologia de maneira geral é mais difícil. O desenvolvimento de ontologias é uma tarefa que pode envolver diversos participantes, de diferentes áreas e com *backgrounds* possivelmente diferentes.

- **Confiabilidade:** A Confiabilidade diz respeito a até que ponto os dados e a análise são dependentes dos pesquisadores. As principais ameaças quanto a confiabilidade

são em relação às medidas utilizadas nos questionários e a quantidade de participantes. Na aplicação dos questionários, o pesquisador estava disponível em caso de dúvidas quanto alguma pergunta presente no questionário. Entretanto, perguntas de questionários são subjetivas e devido a possíveis erros no entendimento, algumas respostas podem ser incorretas. Por fim, a quantidade de participantes pode ser uma ameaça à confiabilidade. No Experimento 2, apenas cinco estudantes participaram da execução o que pode reduzir a capacidade de generalização dos resultados obtidos e a capacidade de descobrir novas informações a partir dos dados coletados.

- **Validade de Construção:** Validade de Construção tem foco na generalização dos resultados em relação a teoria ou conceito do experimento. Nesse contexto, a validade de construção possui duas maiores ameaças. Será que as perguntas realizadas no questionário são boas métricas para validar a facilidade? Além disso, os participantes podem se sentir intimidados na realização do experimento dado que a aplicação dos experimentos foi realizada durante as disciplinas e o professor responsável participou da execução. Desde o início dos experimentos foi explicitado que suas participações não teriam ônus na disciplina ministrada e informações sensíveis (nome e sexo) na realização dos experimento foram omitidas, portanto esta ameaça provavelmente foi minimizada.

4.3 RESULTADOS

4.3.1 Experimento 1

4.3.1.1 Parte 1

O objetivo da parte 1 do experimento foi verificar a compreensão dos usuários participantes do experimento quanto a percepção do raciocínio que ocorreria, com a modelagem dos fragmentos de bases de conhecimento modeladas como ontologias, e também a verificação da facilidade do aprendizado de raciocínio automático.

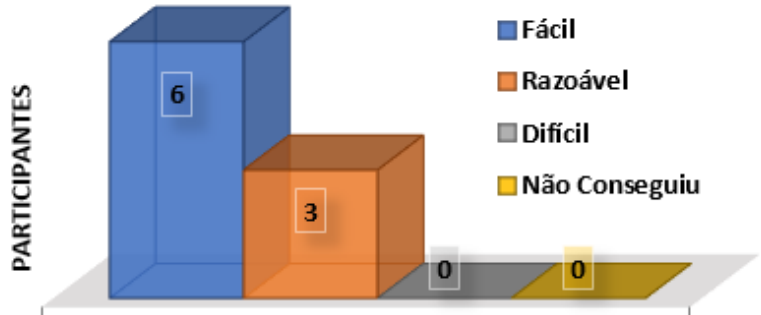
Utilizamos quatro formas distintas de apresentar os mesmos fragmentos ao longo do ensino de Ontologias para turma de Inteligência Artificial da UAG-UFRPE no período de 2018.1. As formas foram:

- O `Medina`;
- O `Protégé`;
- A notação em DL;
- Códigos OWL DL 2.

Usamos um total de trinta e duas horas de ensino de modelagem de bases de conhecimento como OWL e Raciocínio Automático. Antes das trinta e duas horas apresentamos

aos alunos o método *Tableaux* de prova utilizado na máquina de inferência `Pellet`, da qual usamos como componente em nossa arquitetura (Vide Seção 3.1 do Capítulo 3) por doze horas aula. Verificamos também com os participantes do experimento, qual a melhor notação para verificar e aprender sobre raciocínio automático. Ao final de todas as aulas aplicamos o questionário 1, disponível em A.1, no Apêndice A. Apresentaremos a seguir nossos resultados nas subseções a seguir e uma pequena análise qualitativa em cima dos resultados obtidos.

Para cada uma das notações apresentadas aos participantes desta parte do experimento, foi solicitado que os usuários classificassem o nível de percepção de possíveis raciocínios de subsunção ou inconsistência, sem o uso de motores de raciocínio para indicar onde ocorreram realmente os raciocínios. Foram obtidos os seguintes resultados:



	Opção Escolhida
Fácil	6
Razoável	3
Difícil	0
Não Conseguiu	0

Gráfico 1 – Respostas dos usuários sobre o nível de percepção do possível raciocínio automático de Subsunção e Inconsistência no `Medina`

Seis dos nove participantes acham que é fácil perceber onde ocorrerá subsunção e inconsistência, alguns dos participantes informaram que visualmente é mais fácil saber onde ocorrerá inconsistência na base modelada e onde ocorrerá subsunção e mesmo com a máquina de inferência desabilitada, acompanhando apenas o modelo visual. Quando habilitada a máquina de inferência os participantes indicam ainda mais facilidade no aprendizado de ontologias e raciocínio usando o `Medina`, pois são apresentados os raciocínios de inconsistências, os raciocínios de subsunção e as *explanations* em tempo produção, portanto é possível analisar o por que ocorre o raciocínio naqueles pontos determinados e possivelmente não errar mais.

Um ponto negativo relatado por um dos participantes foi:

- "*quando o modelo visual cresce bastante, o usuário fica um pouco perdido e isso começa a dificultar a possível percepção de que o raciocínio pode ocorrer*".

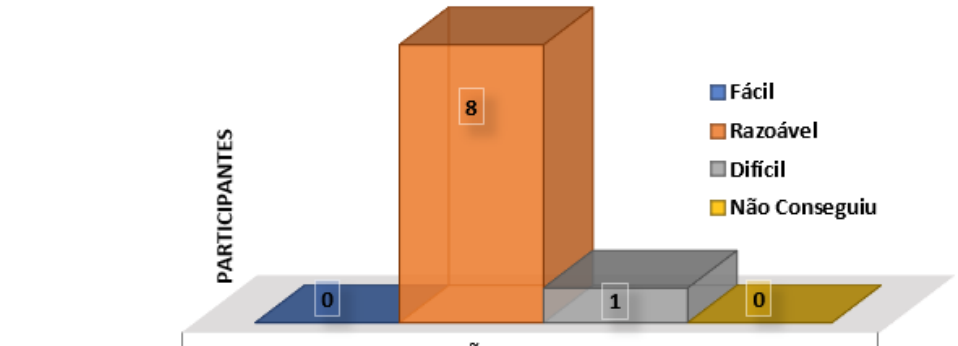
Felizmente a ferramenta apresenta ao usuário de forma automática os raciocínios ocorridos, ou seja, os usuários não precisam se preocupar se vai ocorrer ou não o raciocínio, a ferramenta realiza essa tarefa de forma automática. Ressaltamos que para o aprendizado de alunos e de equipes de desenvolvimento em treinamento essa percepção é relevante, mas na atividade de desenvolvimento real, os usuários não precisam se preocupar em saber se ocorrerá ou não raciocínio.

Outra crítica apresentada foi:

- "*Quanto a explicação dada na ocorrência do raciocínio é difícil e não é intuitiva*".

O `Medina` utiliza a *explanation* dada pela máquina de inferência `Pellet`. Isso pode ser realmente melhorado e usarmos linguagem natural para as *explanations*.

Em relação ao nível de percepção do aluno em descobrir e verificar antecipadamente onde ocorreria o raciocínio automático de subsunção e Inconsistência, dado o modelo desenvolvido em notação DL, obtivemos os seguintes resultados:



	OPÇÕES ESCOLHIDAS
Fácil	0
Razoável	8
Difícil	1
Não Conseguiu	0

Gráfico 2 – Respostas dos usuários sobre o nível de percepção do possível raciocínio automático de Subsunção e Inconsistência na notação DL

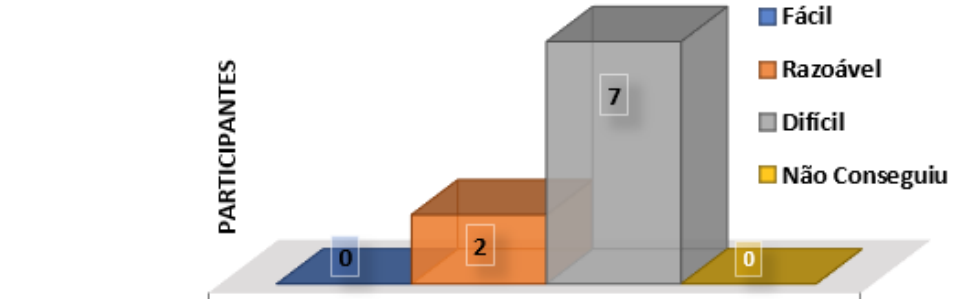
Oito participantes acham razoável verificar a possível ocorrência de raciocínio usando a mesma base de conhecimento, porém modelada em notação DL. Um dos participantes relata que é difícil verificar qualquer ocorrência de raciocínio. Realmente não é intuitivo que vendo um modelo de base de conhecimento em notação DL saber onde ocorrerá raciocínio de subsunção ou onde ocorre inconsistência. Fazendo uma análise e discussão minuciosa em cima do modelo em DL e com algum tempo disponível os usuários conseguem descobrir alguns raciocínios, mas não todos, isso evidencia que o raciocínio realizado em bases de conhecimento deve ser automático e com pouca ou nenhuma intervenção humana, e isso o `Medina` proporciona.

Alguns dos entrevistados relatam que:

- "*O modelo em notação DL não é explicito o suficiente, diferente do modelo visual*".

- "*Embora a notação em DL seja fácil de aprender, não é fácil de verificar a ocorrência de possível raciocínio, principalmente se a base for extensa*".
- "*Não fica claro que um conceito descrito em uma sentença está ligado a outro descrito em outra sentença, o que dificulta saber se ocorrerá ou não raciocínio*".
- "*Verificar inconsistência na base é complicado pois o processo é manual, o aprendizado do raciocínio fica melhor e mais efetivo com o modelo visual apresentado no Medina*".

Em relação ao nível de percepção do aluno em descobrir e verificar antecipadamente onde ocorreria o raciocínio automático de subsunção e inconsistência, dado o modelo desenvolvido em código OWL DL 2, obtivemos os seguintes resultados:



	OPÇÕES ESCOLHIDAS
Fácil	0
Razoável	2
Difícil	7
Não Conseguiu	0

Gráfico 3 – Respostas dos usuários sobre o nível de percepção do possível raciocínio automático de Subsunção e Inconsistência em OWL DL

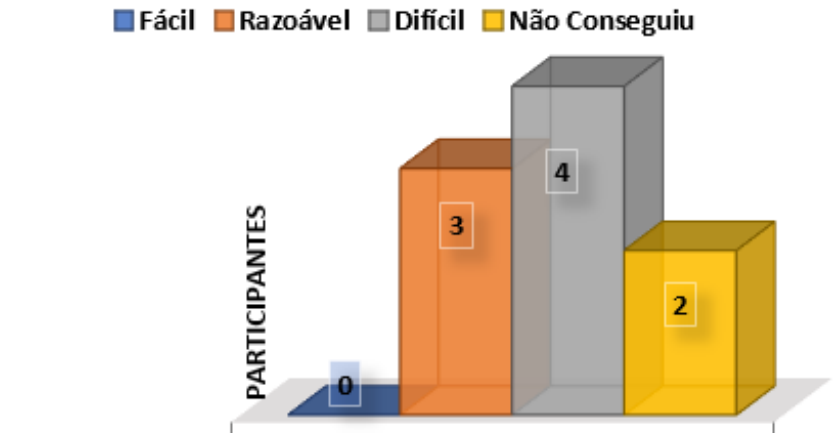
Já esperávamos os resultados obtidos e apresentados no Gráfico 3. Sete dos nove participantes acham uma tarefa não trivial saber onde ocorrerá raciocínio automático analisando apenas código OWL DL 2.

Ensinar o código OWL e raciocínio automático a partir dele não é uma estratégia correta. Os usuários não conseguem perceber onde pode ocorrer o raciocínio. Nenhum dos participantes acha fácil perceber e fazem críticas como:

- "*A representação da base de conhecimento fica extensa e com excesso de conteúdo, pois o código OWL possui uma sintaxe que exige isso e dificulta a leitura e a possibilidade de prever raciocínios*".
- "*Muito código de baixo nível o que polui a base e deixa o entendimento difícil*".
- "*Embora detalhado é mais difícil do que as outras formas de apresentação*".

Ferramentas de Ontologias, visuais ou não, devem prover além da geração de código OWL, modelos mais intuitivos daquele código para o seu aprendizado. Ensinar e aprender apenas com código OWL DL não é uma boa prática. Desenvolvedores e Engenheiros devem conhecer a sintaxe do OWL para determinadas discussões, mas não precisam implementar bases de conhecimento escrevendo o código OWL.

Em relação ao nível de percepção do aluno em descobrir e verificar antecipadamente onde ocorreria o raciocínio automático de subsunção e Inconsistência, dado o modelo desenvolvido na ferramenta `Protégé`, obtivemos os seguintes resultados:



OPÇÕES ESCOLHIDAS	
Fácil	0
Razoável	3
Difícil	4
Não Conseguiu	2

Gráfico 4 – Respostas dos usuários sobre o nível de percepção do possível raciocínio automático de Subsunção e Inconsistência no `Protégé`

Três participantes acharam razoável verificar onde ocorrerá o raciocínio na base de conhecimento usando o `Protégé` e quatro dos nove participantes acham difícil. Nenhum deles achou fácil, isso indica que modelos visuais que representam conhecimento são mais intuitivos e facilitam o aprendizado de ontologias e raciocínio. Mesmo o `Protégé` sendo uma ferramenta amplamente utilizada na comunidade de ontologias e Web Semântica, os participantes do experimento acham difícil entender onde ocorreria raciocínio, o que não aconteceu com uma ferramenta que apresenta as descrições de conceitos (Bases de Conhecimento) de forma visual. Aqui estamos avaliando o quesito facilidade de verificar a ocorrência de raciocínio automático, em outros quesitos o `Protégé` pode ser melhor. Alguns participantes informaram alguns motivos por suas escolhas, vejamos:

- "*A ferramenta é confusa visualmente*".
- "*É uma ferramenta mais complicada em sua apresentação do que as demais*".

- "*O Protégé não é dinâmico, sua interface é complexa e deve-se ter um bom conhecimento para inserir painéis que exibem o raciocínio*".
- "*A ferramenta não é intuitiva, requer um esforço para verificar as subsunções*".

Dois participantes não perceberam que o `Protégé` estava com a máquina de inferência funcionando e realizando raciocínio automático. Comparando com o `Medina` que apresenta o raciocínio em tempo de desenvolvimento e de forma visual, não há dúvidas de ser uma forma melhor de aprender e verificar possíveis ocorrências de erros no desenvolvimento de ontologias em OWL 2 DL e deduções automáticas. O uso do Protégé só foi melhor que analisar puramente o código OWL, inclusive pelos resultados observamos que os participantes acharam melhor a notação DL do que o `Protégé`, pelo menos para o aprendizado de raciocínio automático.

4.3.1.2 Parte 2

Pedimos que os participantes do experimento fizessem um ranking (Verificar questionário na Seção A.1 do Apêndice A) da melhor a pior forma de verificar e aprender raciocínio automático com as notações e ferramentas apresentadas, pontuamos o ranking da seguinte forma, a escolha número um, recebe 5 pontos, a escolha dois, 4 pontos, a escolha 3, 3 pontos, a escolha quatro 2 pontos e a última escolha, 0 pontos. Os resultados obtidos são apresentados no Gráfico 5:

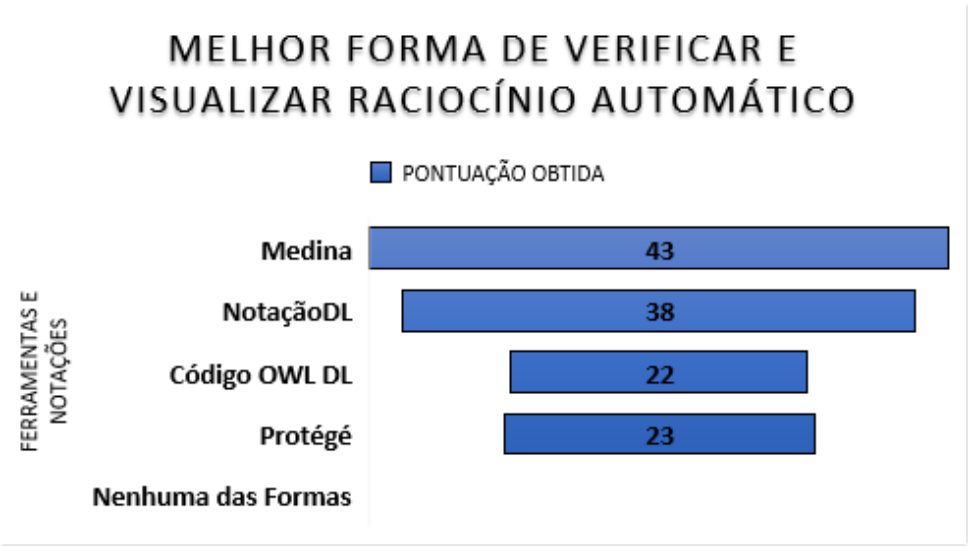


Gráfico 5 – *Ranking* dos usuários das melhores formas de verificar o raciocínio automático ocorrido nos fragmentos

O `Medina` foi a ferramenta escolhida em primeiro lugar no ranking, apenas 2 participantes consideraram o `Medina` como segunda opção de uso, optando ambos pela notação DL. O `Protégé` foi no ranking final a terceira opção, perdendo para a notação em DL que

é totalmente manual. Alguns participantes justificaram suas escolhas com os seguintes argumentos:

- "A ferramenta visual facilita a visualização e é a primeira opção, talvez com mais uso do Protégé o mesmo se torne preferência em relação a notação em DL".
- "Ter auxilio visual, especialmente quando se tem muita informação modelada, deixa a verificação do raciocínio mais simples".

Os dois participantes que ranquearam `Medina` como segunda opção informaram que a notação em DL é a sua primeira opção e afirmaram que:

- "*A notação em DL e visual com o* `Medina` *são mais intuitivas*".
- "*Possuo maior afinidade com a notação em DL*".

Verificamos que poderíamos em trabalhos futuros, associar o modelo visual com a notação em DL, tal como uma associação apresentando aos usuários da ferramenta as duas notações. Não é surpresa também que verificar possível raciocínio analisando código puramente em OWL DL é a pior opção. Apresentamos no Gráfico 6 os resultados do ranking para melhor forma de aprender sobre raciocínio automático.

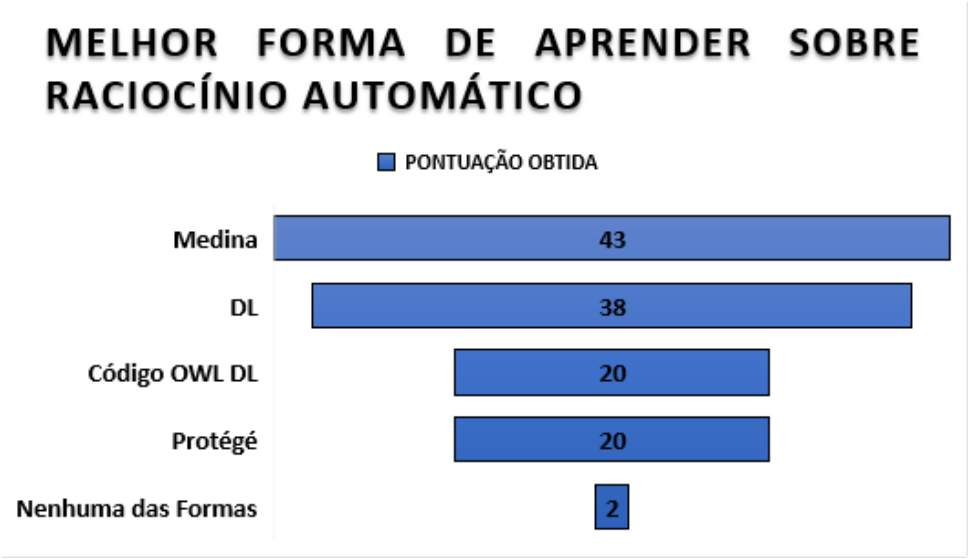


Gráfico 6 – *Ranking* dos usuários das melhores formas de aprender sobre o raciocínio automático ocorrido nos fragmentos

O `Medina` foi novamente a ferramenta escolhida em primeiro lugar no ranking, apenas 2 participantes consideraram o `Medina` como segunda opção de uso, optando ambos pela notação DL. O`Protégé` empatou no ranking final em terceira opção com o código OWL DL, um dos participantes votou no `Protégé` em última opção, esse voto, infelizmente não sabemos se foi erro ou de fato opção do participante, porém, temos que considerar como a opção escolhida e não podemos alterar os resultados. Mais uma vez a notação em DL foi a

opção em segunda posição no ranking de aprendizado de raciocínio automático. Nenhum dos participantes justificou suas escolhas.

4.3.1.3 Parte 3

Na terceira parte do Experimento 1 apenas seis dos nove estudantes que participaram nas partes anteriores se disponibilizaram para a execução dessa parte do experimento. Os participantes foram separados em dois grupos, com três estudantes cada grupo. Ambos os grupos desenvolveram o mesmo conjunto de axiomas utilizando o `Medina` e o `Protégé` (a ontologia O_0 apresentada na Seção 4.2.2 desse capítulo). O objetivo desta parte do experimento foi verificar como os usuários modelam a ontologia O_0, de que forma os integrantes do grupo interagem no desenvolvimento, quais práticas são realizadas e em quanto tempo a ontologia é desenvolvida.

Na primeira rodada da execução da parte 3 do Experimento 1, o Grupo 1 desenvolveu a ontologia O_0 no `Medina` e simultaneamente o Grupo 2 desenvolveu a ontologia O_0 no `Protégé`, sendo possível dessa forma verificar a interação dos integrantes do grupo com cada uma das ferramentas. Finalizado o desenvolvimento para ambos os grupos, foi realizada a segunda rodada invertendo a ferramenta utilizada na modelagem: Grupo 2 desenvolveu a ontologia O_0 no `Medina` e o Grupo 1 desenvolveu a ontologia O_0 no `Protégé` verificando novamente a interação dos integrantes do grupo, entre si e com a ferramenta utilizada.

A seguir são apresentadas a análises qualitativa e discussões sobre os resultados obtidos na parte 3 do Experimento 1.

4.3.1.3.1 Análise Qualitativa

Ao fim do desenvolvimento do conjunto de axiomas nas ferramentas, foi analisado o tempo de desenvolvimento da ontologia O_0 para cada um dos grupos.

Tabela 6 – Tempo de modelagem dos Grupos 1 e 2 na execução da parte 3 do Experimento 1. * Experimento foi interrompido no tempo apresentado pois o `Protégé` parou de funcionar.

Ferramenta	Grupo	Tempo (mm:ss)
`Medina`	Grupo 1	15:37
`Protégé`	Grupo 1	09:58
`Medina`	Grupo 2	18:35
`Protégé`	Grupo 2	15:14*

Apresentamos na Tabela 6 os tempos de desenvolvimento da ontologia O_0 pelos Grupos 1 e 2 no Experimento 1. O Grupo 1 modelou o conjunto de axiomas em 15 minutos e 37 segundos na ferramenta `Medina` e em 9 minutos e 58 segundos na ferramenta `Protégé`.

Nesse caso o tempo de desenvolvimento no `Protégé` foi 5 minutos e 39 segundos menor que no `Medina`. O Grupo 2 desenvolveu a ontologia em 18 minutos e 35 segundos na ferramenta `Medina`. O experimento do Grupo 2 na ferramenta `Protégé` foi interrompido pois a mesma parou de funcionar aos 15 minutos e 14 segundos não sendo possível obter o tempo de desenvolvimento final. Isso é uma falha corriqueira do Protégé, o que leva a perda de tempo e produtividade das equipes de desenvolvimento.

O Grupo 2 modelou a ontologia O_0 no `Medina` e no `Protégé` em mais tempo que em relação ao Grupo 1. Mesmo apresentando problemas na modelagem no `Protégé`, o desenvolvimento do Grupo 2 nessa ferramenta já ultrapassava 5 minutos e 16 segundos a mais em relação ao Grupo 1. Devido ao nível de experiência equivalente entre os integrantes do Grupo 2, foram percebidos conflitos na decisão de como os axiomas deveriam ser construídos na ferramenta.

O Grupo 1 obteve um tempo de modelagem menor que o Grupo 2 no desenvolvimento da ontologia O_0 no `Medina` (2 minutos e 58 segundos). Verificando a modelagem visual de ambos os grupos foi percebido que a disposição dos elementos visuais estava bem diferente. No Grupo 1, os elementos visuais estavam mais espaçados entre si e agrupados de acordo com suas ligações (Vide Figura 44) o que facilita encontrar quais classes e propriedades já estão construídas e conectá-las.

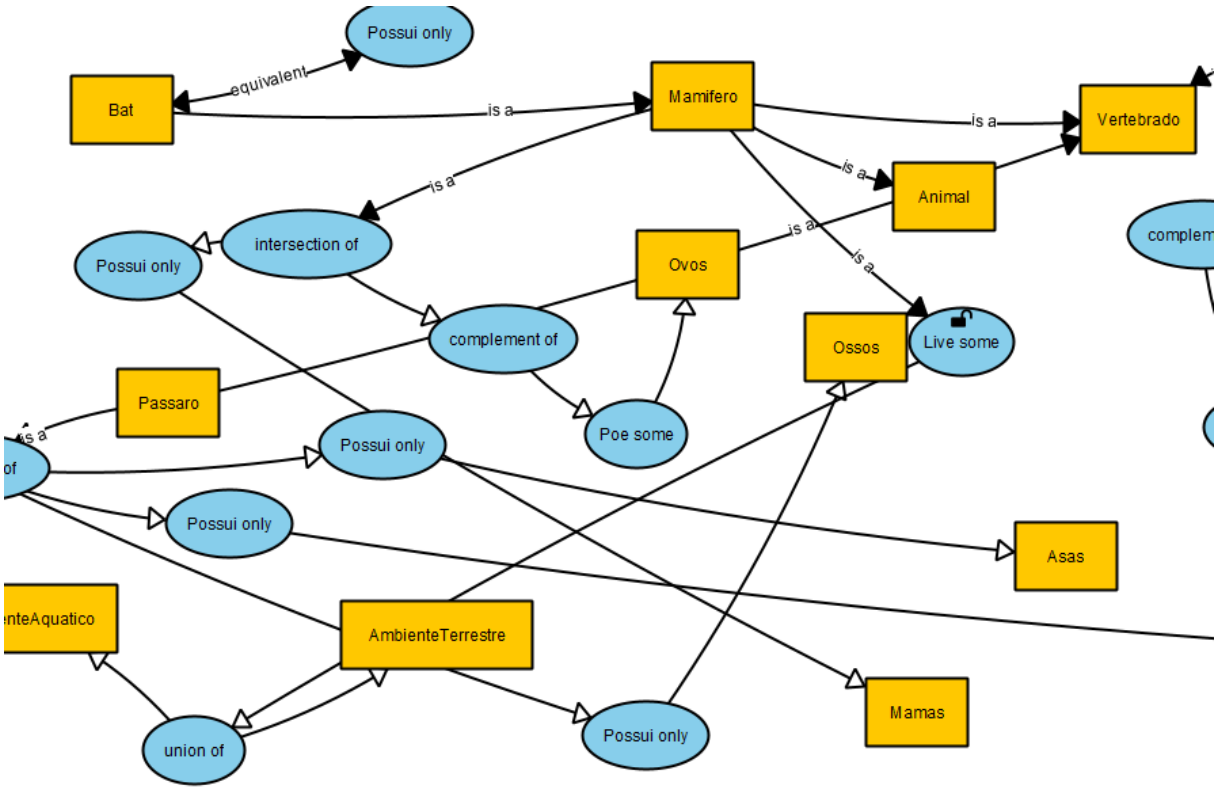


Figura 44 – Fragmento da modelagem visual do Grupo 1 no `Medina`

Diferentemente do Grupo 1, o Grupo 2 não focou na organização dos elementos visuais (Vide Figura 45) gerando uma poluição visual e dificultando a busca por elementos visuais

existentes, podendo acarretar em tentativas de adição duplicadas de elementos visuais já existentes ou a dificuldade em conectar os elementos visuais necessários para a construção dos axiomas. Dessa forma, foi percebido que a organização da modelagem e a forma como o grupo interagiu são qualidades essenciais no desenvolvimento de uma ontologia e influenciam em seu tempo de modelagem.

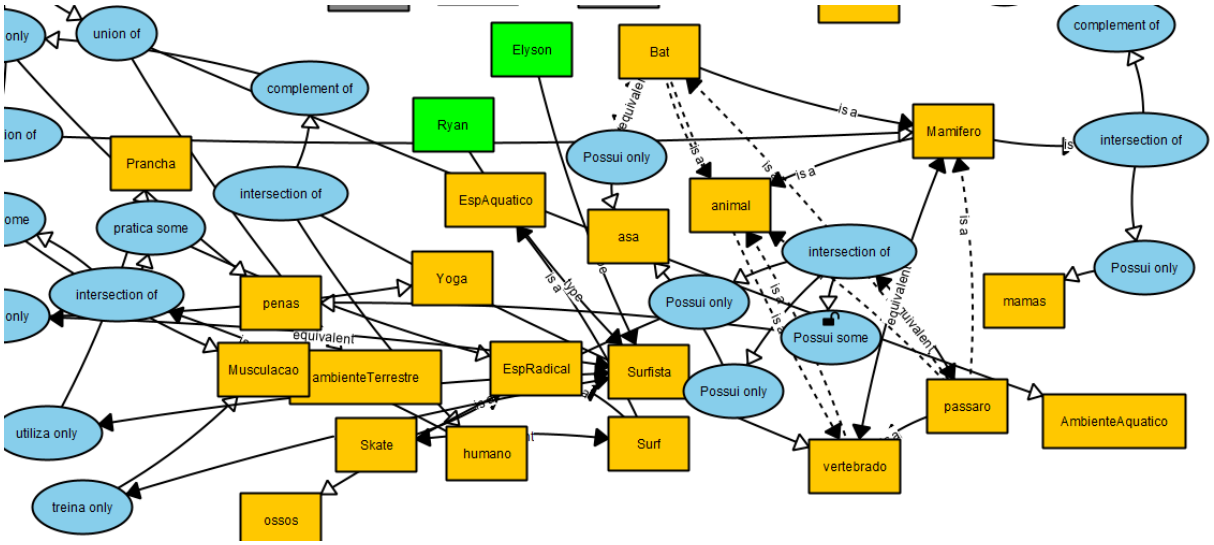


Figura 45 – Fragmento da modelagem visual do Grupo 2 no `Medina`

4.3.1.3.2 Discussão dos Resultados

Foi percebido neste experimento que:

- Criar primeiro todas as classes e propriedades da ontologia não é uma boa prática com uso do `Medina`.

  Discussão:

  Criar todas as classes e propriedades da ontologia não é tão efetivo quanto parece. Ao criar todos esses elementos visuais, o usuário acaba não estabelecendo uma organização entre as classes e propriedades. Quando necessária a criação de arcos entre os elementos visuais, se o usuário não reorganizar a posição dos nós participantes do arco, terá dificuldade na compreensão da representação visual desenvolvida.

- Foi possível identificar também uma influência da organização dos elementos visuais com a produtividade dos grupos neste experimento.

  Discussão:

  Isso mostra que é necessário estabelecer boas práticas na criação dos axiomas e que pode ser necessário algum treinamento prévio para a utilização do `Medina`, dado que a abordagem do sistema proposto não contribuiu com a intuitividade em alguns momentos do desenvolvimento.

> Uma possível boa prática é a construção da ontologia a partir de seus axiomas, criando apenas as classes e propriedades relacionadas ao axioma por vez. Além disso, na criação das classes e propriedades, pode ser uma boa prática já estabelecer uma organização baseada no axioma em desenvolvimento, distribuindo as classes e propriedades em posições que permitam a conexão entre os elementos visuais de maneira bem organizada.

Com a execução do Experimento 1, foi percebido também que o tempo de construção da ontologia estava diretamente influenciado pela participação dos integrantes do grupo. No Grupo 1, todos os integrantes participaram ativamente do desenvolvimento, diferentemente do Grupo 2, em que um dos integrantes não participou ativamente do desenvolvimento.

Além disso, foi percebido que o desenvolvedor com mais habilidade ser o principal usuário da ferramenta, tanto no `Medina` como no `Protégé` influenciou positivamente no tempo da construção da ontologia. No Grupo 1 o desenvolvedor com mais habilidade foi o operador das ferramentas enquanto no Grupo 2 isso não ocorreu. O Grupo 1 obteve tempo de execução melhor (2 minutos e 58 segundos a menos) que o Grupo 2, o que talvez possa ser justificado pela participação dos integrantes do grupo e o uso do desenvolvedor mais habilidoso como usuário principal. Outra observação é que no Grupo 2 todos possuíam a mesma experiência e isso gerava conflitos na hora da modelagem, talvez utilizando das boas práticas de programação em pares, mesclando as equipes em experientes e menos experientes resolva o problema.

4.3.2 Experimento 2

Participaram do Experimento 2 cinco alunos, cada um, individualmente desenvolvendo as ontologias O_1 e O_2 utilizando o `Protégé` e o `Medina`. O objetivo deste experimento foi verificar como os participantes desenvolvem, individualmente, ontologias no `Medina` e no `Protégé`, levando em consideração possíveis boas práticas, organização visual, corretude dos axiomas inseridos e o tempo necessário para a construção das ontologias. A seguir apresentamos nossas análises quantitativa e qualitativa referentes ao Experimento 2.

4.3.2.1 Análise Quantitativa

No Experimento 2 foram obtidos vídeos[2] da modelagem de cada usuário, através de *software* de captura de tela, com o objetivo de compreender como os usuários realizam a modelagem, organizam os elementos visuais e em quanto tempo desenvolvem a ontologia no `Medina` e no `Protégé`. Ao analisar os vídeos obtidos da modelagem dos usuários nas ferramentas, foram extraídos os seguintes dados:

[2] Link do repositório com os vídeos: <https://goo.gl/CBjWzk>

- Nomes incorretos: quando algum dos usuários inseriu um nome de classe ou propriedade que não corresponde ao nome apresentado no axioma, incluindo abreviações.
- Percentual de corretude da modelagem da ontologia: percentual de corretude da modelagem dos axiomas presentes na ontologia. Para calcular o percentual foi levada em consideração uma pontuação para o desenvolvimento de cada axioma da ontologia de acordo com a tabela a seguir:

Pontuação	Descrição	Exemplo
3	O axioma foi inserido corretamente	$A \sqsubseteq B \sqcup C$
2	O axioma possui todas as classes e propriedades corretamente, mas possui erros lógicos na construção do mesmo	$A \sqsubseteq B \sqcap C$
1	O axioma possui classes ou propriedades incorretas ou ainda alguma entidade está ausente	$A \sqsubseteq B1 \sqcup C$
0	O axioma não foi inserido	

- Tempo de modelagem: tempo gasto pelo usuário para desenvolver a ontologia inteira e verificar as inferências e possíveis insatisfatibilidades decorrentes da construção dos axiomas.

Tabela 8 – Dados obtidos da análise dos vídeos da modelagem dos usuários nas ferramentas `Medina` e `Protégé` agrupados por ferramenta utilizada.

Ferramenta	Nomes incorretos (média)	Corretude (média)	Tempo (média)
`Medina`	$4,4$ por ontologia	$84,85\%$	32:35
Protege	$9,8$ por ontologia	$84,38\%$	26:10

Visualizamos na Tabela 8 as médias de todos os usuários da quantidade de nomes incorretos, corretude e tempo de modelagem. A quantidade de nomes incorretos é consideravelmente maior na ferramenta `Protégé` em relação ao `Medina` (9,8 e 4,4 respectivamente). Essa diferença pode ser atribuída às abreviações dos nomes corretos que, por questões de erro no `Protégé` ou por questões de eficiência na digitação de nomes consideravelmente grandes[3], foram utilizadas. Já a corretude na modelagem da ontologia foi um pouco melhor na ferramenta `Medina` que na ferramenta `Protégé`, com uma diferença de 0,47%. Quanto ao tempo da modelagem, na ferramenta `Medina` o tempo foi maior em 6 minutos e 25 segundos em relação ao `Protégé`. A modelagem visual é uma forma diferenciada de desenvolver ontologias, uma vez que os usuários sempre organizavam as entidades visuais, conferindo sua corretude, verificando as inferências apresentadas visualmente e consequentemente modelando a ontologia em mais tempo. Além disso, a quantidade

[3] *HealthSurveillanceNotificationAction*, por exemplo

consideravelmente maior de nomes abreviados ao modelar a ferramenta `Protégé` no Experimento 2 influencia positivamente em seu tempo de desenvolvimento, uma vez que com nomes menores é possível construir um axioma mais rápido, possivelmente influenciando nos tempos de desenvolvimento.

4.3.2.2 Análise Qualitativa

Cada usuário modelou as ontologias O_1 e O_2 (Vide Seção 4.2.2) na realização do Experimento 2. Ao término da modelagem de cada ontologia (O_1 e O_2), foram aplicados dois questionários. No primeiro questionário (Vide Seção A.2 do Apêndice A) foi solicitado que os participantes informassem quais foram as suas avaliações a respeito de funcionalidades específicas da ferramenta utilizada na construção das ontologias (`Medina` ou `Protégé`). No segundo questionário (Vide Seção A.3 do Apêndice A) os participantes responderam questões envolvendo opiniões específicas sobre o `Medina`.

Na primeira pergunta foi solicitada a avaliação em relação a encontrar as funcionalidades necessárias para desenvolver a ontologia.

Conforme observado no Gráfico 7, encontrar as funcionalidades necessárias na ferramenta `Medina` foi considerada muito fácil para um usuário (20%) e, de maneira semelhante, um usuário também considerou muito fácil encontrar as funcionalidades necessárias no `Protégé`. Quatro usuários (80%) consideraram fácil encontrar as funcionalidades necessárias no `Medina` e quatro participantes (80%) acharam dificuldade mediana encontrar as funcionalidades no `Protégé`.

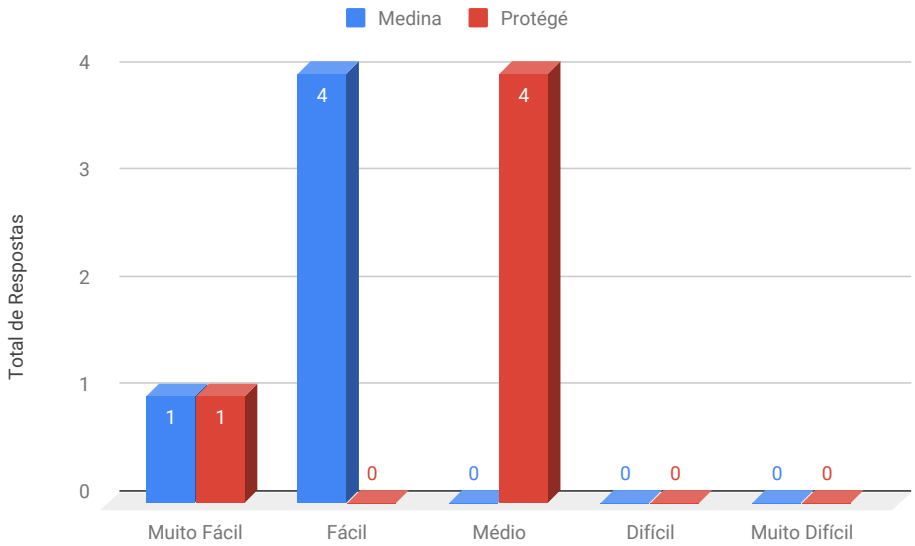


Gráfico 7 – Respostas dos usuários a "Qual a sua avaliação em relação a encontrar as funcionalidades necessárias para modelar a ontologia"

Todas as funcionalidades do `Medina` são apresentadas na tela inicial e não é necessário mudar de abas para obter raciocínios sobre uma ontologia em desenvolvimento, diferen-

temente do Protégé, sendo necessário inclusive iniciar a execução de um raciocinador (máquina de inferência) quando necessário. Além disso o Medina proporciona a seus usuários uma modelagem de classes, propriedades e indivíduos em um único painel, enquanto no Protégé é necessário a navegação em abas. Ressaltamos que não achamos o Protégé uma ferramenta ruim, mas a forma de trabalho que o Medina proporciona é mais simples e rápida.

Em relação a modelagem da ontologia, segunda questão do questionário (Vide Seção A.2 do Apêndice) um usuário (20%) achou muito simples o seu uso para o Medina e um usuário também achou muito simples o seu uso no Protégé (Vide Figura 8). Três usuários (60%) acharam a modelagem no Medina simples enquanto no Protégé apenas um usuário também achou simples. Um usuário (20%) achou mediana a modelagem em ambas as ferramentas e dois usuários (40%) acharam complicada a modelagem de ontologias no Protégé. Importante comentar que nenhuma das avaliações dos usuários a respeito da modelagem da ontologia foi negativa (categorizada como complicado ou muito complicado), diferentemente do Protégé, em que dois participantes do experimento avaliaram a modelagem como complicada.

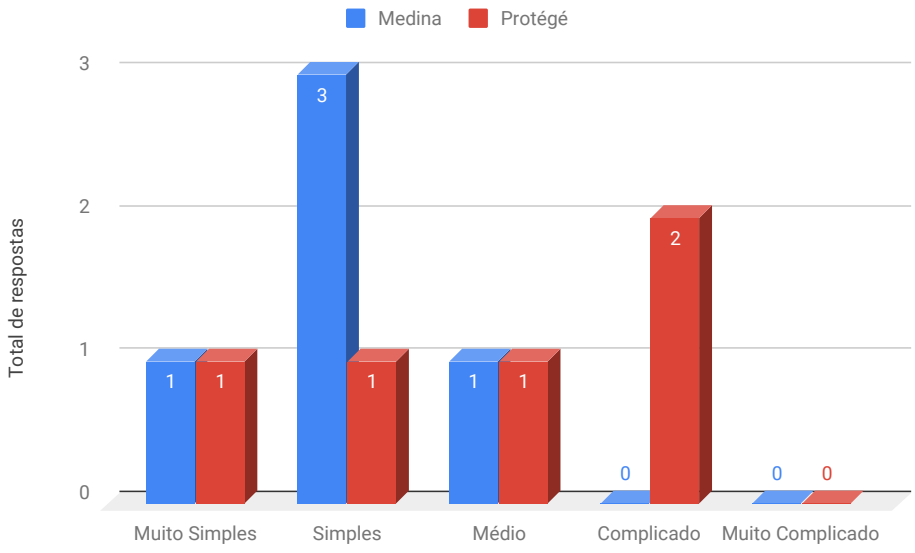


Gráfico 8 – Respostas dos usuários a "Qual a sua avaliação em relação a modelagem da ontologia"

Para cinco dos usuários (100%), a inferência apresentada em pleno desenvolvimento das ontologias é uma funcionalidade muito útil no Medina e para dois usuários (40%) a inferência também é muito útil no Protégé, conforme pode ser observado no Gráfico 9. Para os demais usuários do Protégé, a inferência tem utilidade média (um usuário - 20%), pouca relevância (um usuário - 20%) ou irrelevante (um usuário - 20%).

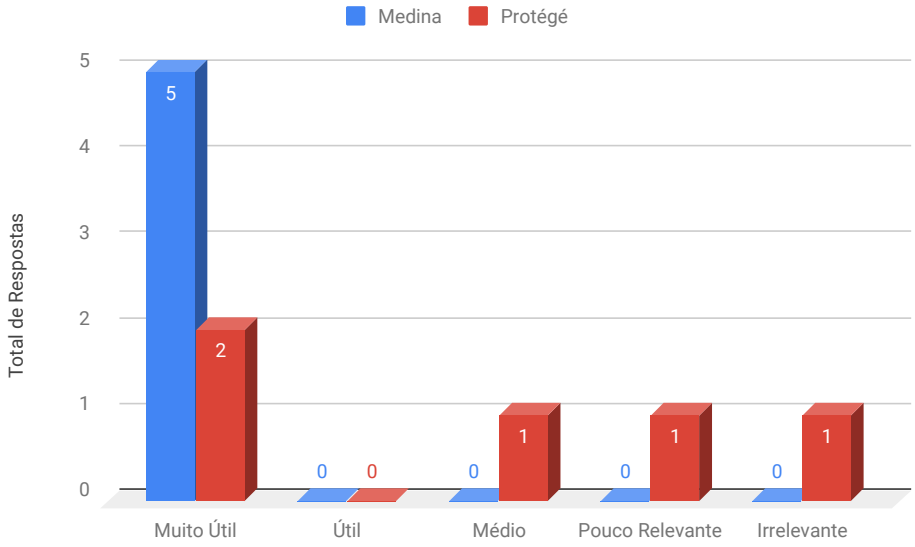


Gráfico 9 – Respostas dos usuários a "Qual a sua avaliação em relação as inferências sobre a ontologia modelada através da ferramenta"

Na terceira questão do primeiro questionário aplicado no Experimento 2, foi solicitada a avaliação do usuário em relação a checagem de insatisfatibilidade (Vide Gráfico 10), outra importante tarefa de raciocínio. Para três usuários (60%), a checagem de insatisfatibilidades do `Medina` é muito útil. Já dois usuários (40%) acharam útil a checagem para as ferramentas `Medina`. Dois usuários (40%) também acharam útil a checagem de insatisfatibilidade no `Protégé`. Dois usuários (40%) acharam a checagem com utilidade média no `Protégé` e um dos usuários (20%) achou pouco relevante também no `Protégé`.

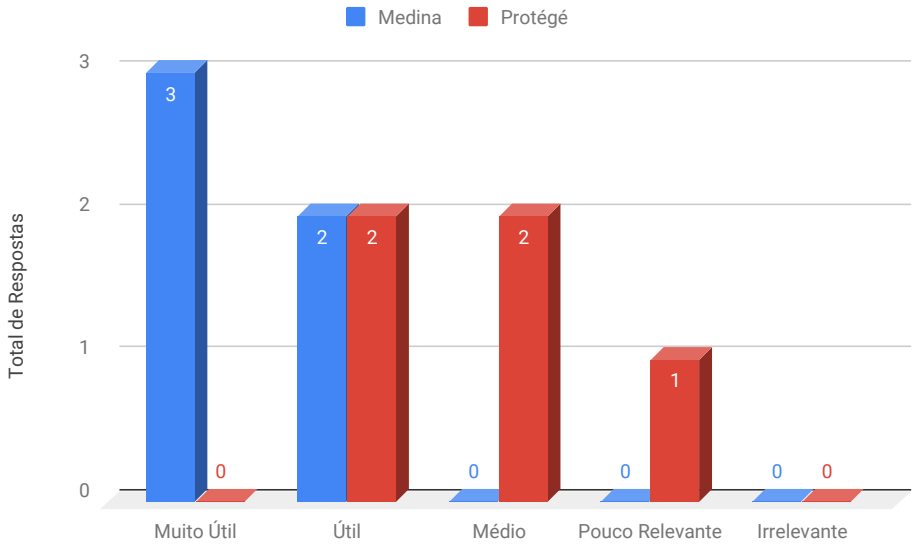


Gráfico 10 – Respostas dos usuários a "Qual a sua avaliação em relação a visualização das insatisfatibilidades de classes sobre a ontologia modelada através da ferramenta"

No segundo questionário (disponível na Seção A.3 do Apêndice), aplicado após o desenvolvimento de cada usuário nas duas ferramentas, as questões estavam relacionadas a satisfação das funcionalidades do `Medina`, sem levar em consideração o `Protégé` ou outra ferramenta.

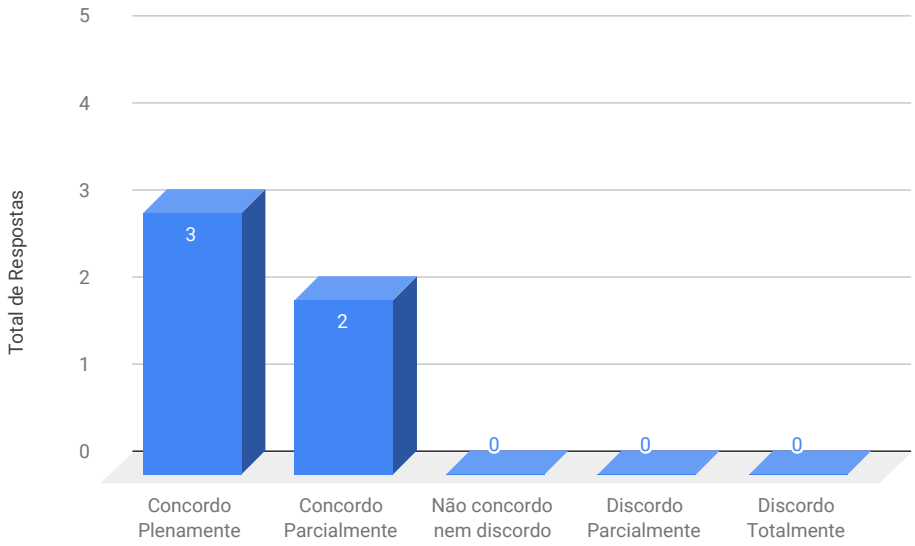


Gráfico 11 – Respostas dos usuários a "A visualização de deduções apresentada pelo `Medina` é adequada para as minhas necessidades."

Três usuários (60%) concordam totalmente que a visualização de deduções do `Medina` é adequada as suas necessidades, enquanto dois usuários (40%) concordaram parcialmente com a afirmação. Os resultados podem ser vistos no Gráfico 11.

A modelagem visual é adequada para às necessidades dos usuários? Foi a segunda afirmação do questionário. Dois usuários (40%) concordaram totalmente com essa afirmação, dois usuários (40%) concordaram parcialmente e um dos usuários (20%) discordou parcialmente quanto a afirmação sobre a modelagem visual. Os resultados dessa questão são apresentados no Gráfico 12.

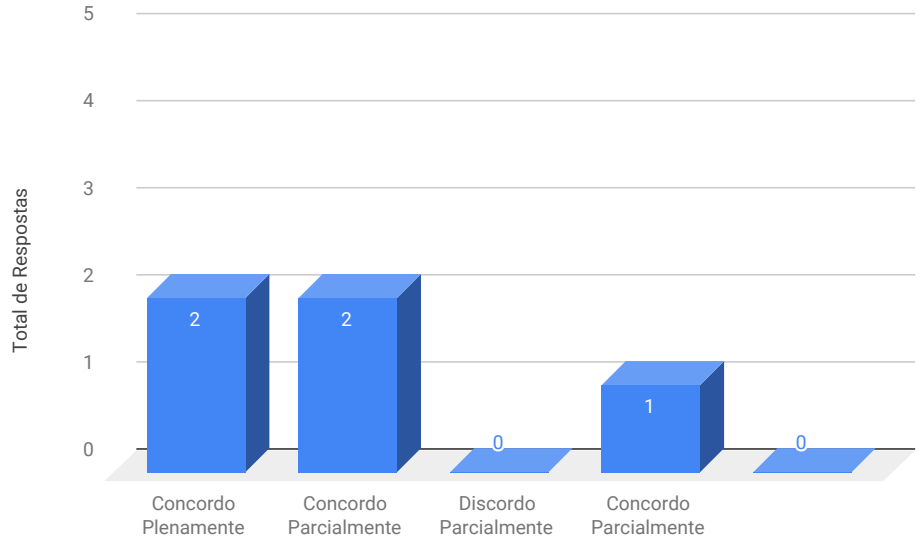


Gráfico 12 – Respostas dos usuários a "A modelagem visual apresentada pelo `Medina` é adequada as minhas necessidades."

Uma das modificações mais relevantes do `Medina` após o Experimento 1 foi a possibilidade de ocultar axiomas complexos. Na afirmação relacionada a ocultar axiomas, os cinco usuários (100%) concordaram totalmente que essa funcionalidade é adequada às suas necessidades, os resultados podem ser observados no Gráfico 13.

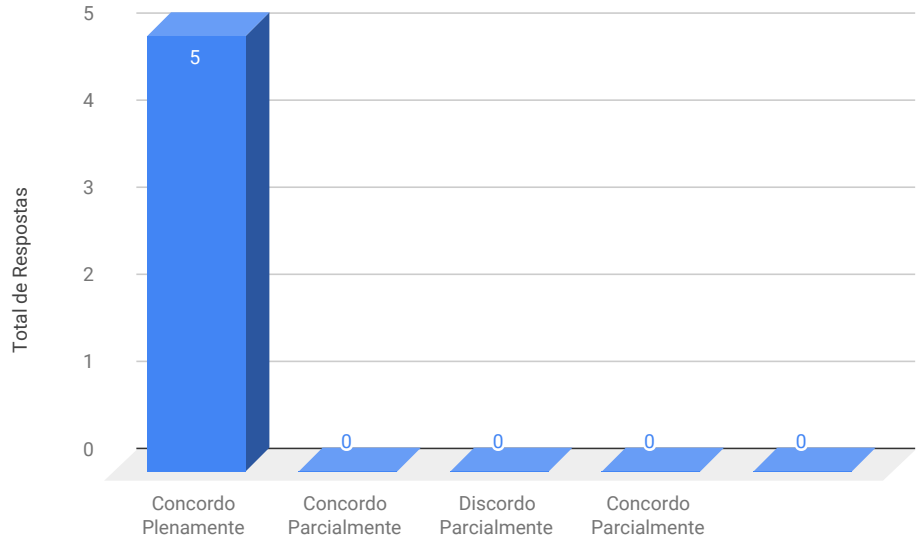


Gráfico 13 – Respostas dos usuários a "A funcionalidade de ocultar axiomas apresentada pelo `Medina` é adequada as minhas necessidades"

Ao fim do desenvolvimento das ontologias O_1 e O_2 (Vide Seção 4.2.2), foram obtidos arquivos gerados pelas ferramentas no formato OWL DL 2 e DR (formato padrão do `Medina`).

A seguir são apresentados alguns fragmentos da modelagem visual feita pelos usuários no `Medina`. Na Figura 46 é possível observar como um dos usuários construiu a ontologia de maneira visual, sendo este o único entre os participantes a utilizar a funcionalidade de adição de axiomas a partir da *Manchester Syntax*. A disposição dos construtores de classe criados no mapeamento da *Manchester Syntax* é baseada na posição das classes já existentes.

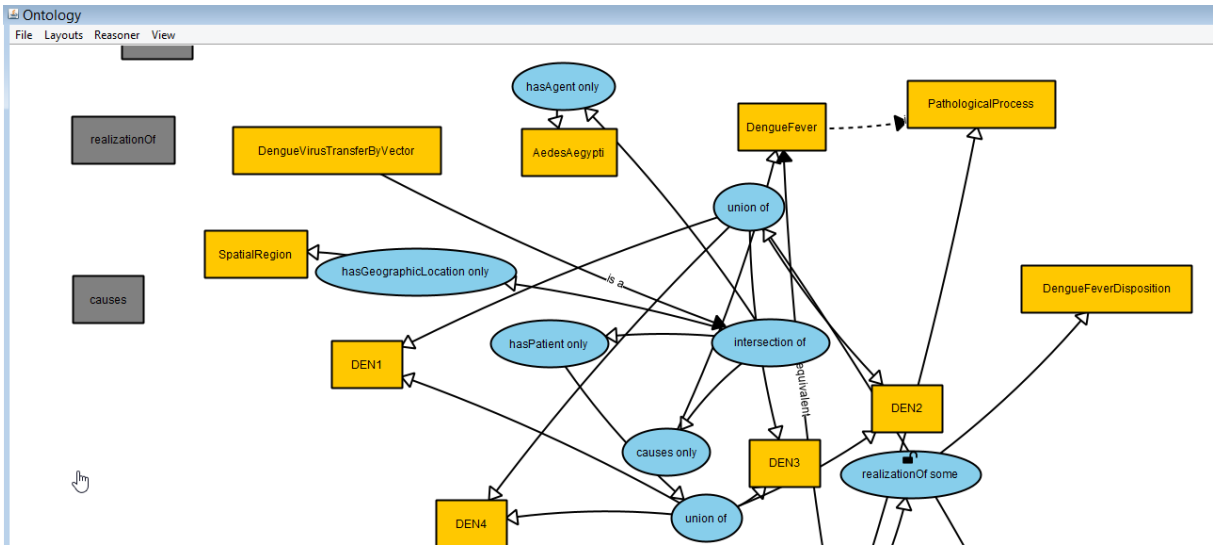


Figura 46 – Fragmento de modelagem visual realizada por um dos usuários no `Medina`

O usuário que modelou o fragmento da ontologia O_2 apresentado na Figura 47, desenvolveu os axiomas a partir da modelagem manual, criando cada um dos construtores e os conectando. É possível observar que o usuário decidiu criar os construtores mais distantes um do outro. E também é possível observar que ele utilizou a funcionalidade de ocultar axiomas, no canto inferior direito da modelagem.

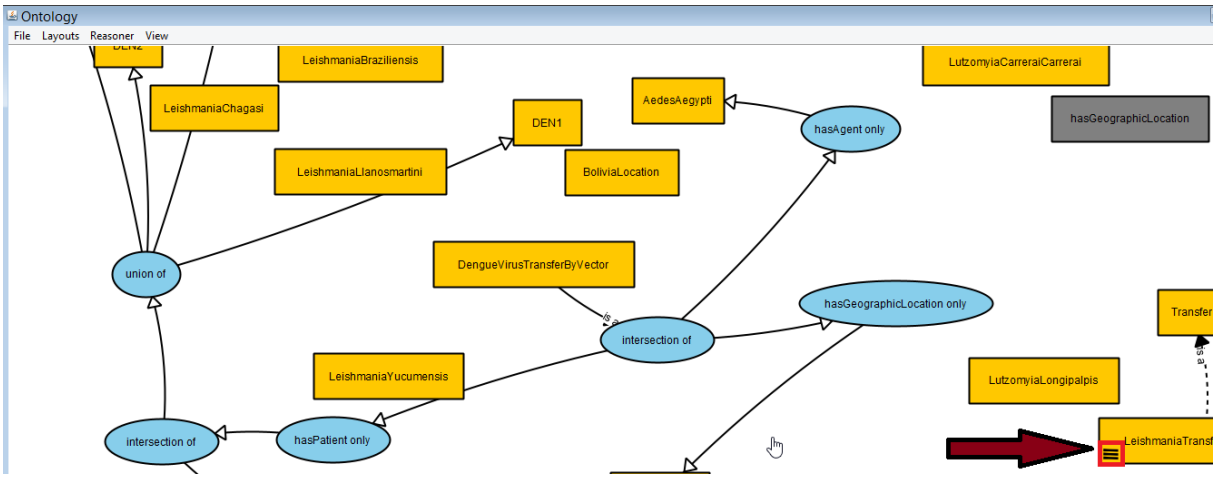


Figura 47 – Fragmento de modelagem visual realizada por um dos usuários no `Medina`

Já um terceiro usuário, conforme pode ser observado na Figura 48, preferiu utilizar a funcionalidade de ocultar axiomas após o desenvolvimento do mesmo e também

a de ocultar classes e propriedades que não eram mais necessárias no desenvolvimento. Consequentemente a modelagem visual desenvolvida por ele possui uma quantidade de elementos visuais menor. Note que as inferências continuam ocorrendo (Vide Figura 48 destacada em vermelho), mesmo quando os axiomas estão ocultos.

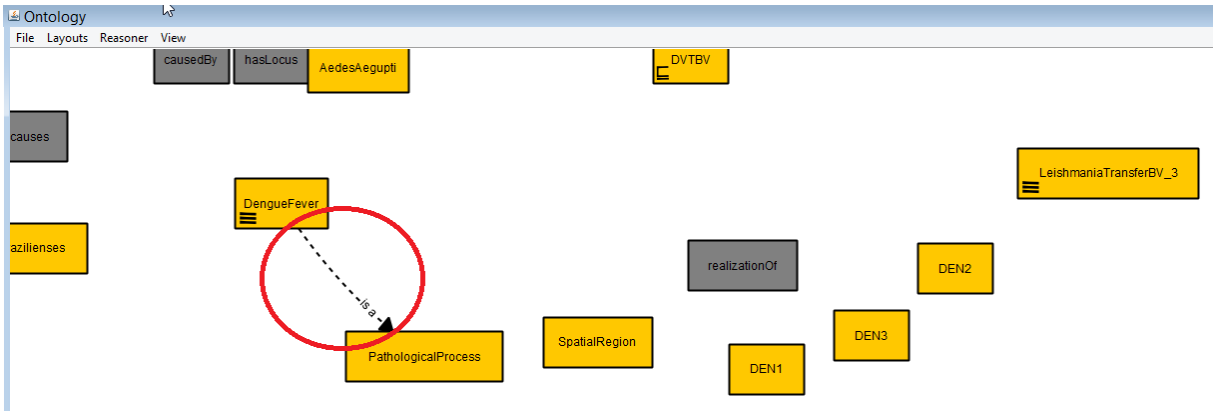


Figura 48 – Fragmento de modelagem visual realizada por um dos usuários no `Medina`

No segundo experimento, entre as perguntas dos questionários, foi pedido para que os usuários adicionassem, de maneira livre, comentários em relação a modelagem em cada uma das ferramentas e que apresentassem pontos positivos e negativos do `Medina` e do `Protégé`.

Os comentários escritos sobre o `Protégé` estão relacionados a um problema que aconteceu em todas as modelagens. Ao inserir palavras grandes, como por exemplo, *HealthSurveillanceNotificationAction* o `Protégé` parava de funcionar, em alguns casos implicando em perda de axiomas que não foram salvos anteriormente.

Já em relação a ferramenta `Medina` temos como pontos negativos a falta de atalhos; a ausência da funcionalidade de completar palavras, semelhante ao `Protégé`, e a possibilidade de "Poder andar pela tela arrastando para ter uma melhor organização". Atualmente existe o atalho de desfazer e refazer, porém os usuários gostariam de mais atalhos para criar classes ou selecionar construtores de maneira mais eficiente. Além disso algumas queixas em relação a barra de rolagem foram verificadas bem como dois erros na inserção de axiomas foram apresentados na modelagem feita pelos usuários.

De comentários positivos, três usuários elogiaram o uso de ocultar axiomas como uma funcionalidade de bastante utilidade para evitar, ou pelo menos minimizar os problemas inerentes da visualização de grafos. A modelagem visual, segundo um dos usuários, é mais didática e autoexplicativa, enquanto outro afirma que é mais fácil de modelar de maneira visual. Dois usuários também comentaram de maneira positiva sobre a visualização de inferências e insatisfatibilidades.

Após os experimentos, todos os vídeos obtidos do desenvolvimento de ontologias feito pelos usuários foram verificados[4] em busca de outras características ou reforçar as opiniões

[4] Por questões técnicas não foi possível obter parte da modelagem no `Protégé` de um dos usuários.

apresentadas pelos usuários. Foi percebido que o uso de deduções apresentado visualmente chamou a atenção dos usuários na modelagem, havendo uma preocupação em verificar o que foi deduzido. Além disso, o uso de ícones com mesmo símbolo que os utilizados em axiomas em DL tiveram bom entendimento por parte dos usuários.

O uso da barra lateral (*Tree View*) que apresenta as propriedades, classes e indivíduos de maneira semelhante ao `Protégé` foi utilizado por todos os usuários, com a finalidade de verificar a existência de classes ou propriedades. O uso de técnicas de visualização diferentes foi bem aproveitado pelos participantes, visto que uma das maiores vantagens da técnica implementada na barra lateral (*Idented List*) é a facilidade de uso, uma vez que a maioria dos usuários de computadores estão familiarizados com a técnica.

Já os construtores existencial ($\exists$) e universal ($\forall$) foram os componentes visuais que os usuários mais tiveram dificuldade em utilizar. Alguns dos usuários inclusive buscavam conectar a propriedade diretamente com a classe, considerando o construtor como um arco, sem criar o construtor existencial ou universal através do painel. Um maior treinamento para os usuários do `Medina`, focando nesses construtores é uma abordagem para minimizar esse problema. Além disso, notações alternativas para a representação dos construtores existencial e universal podem facilitar o uso deles por parte dos usuários no `Medina`.

Por fim, em alguns casos foi percebido dificuldade por parte dos usuários em escrever os axiomas em ambas as ferramentas. Dentre eles, um usuário que apresentou diversas dificuldades em utilizar os construtores tanto de interseção quanto de união no `Medina`, solicitando sempre dicas do observador. A falta de uso e experiência desses usuários são as mais prováveis causas para a dificuldade percebida. Com uma quantidade maior de treinamento e experiência provavelmente minimizam esse problema.

4.3.3 Discussão

Embora o foco dos experimentos tenha sido verificar fatos qualitativos e quantitativos a respeito do `Medina`, sentiu-se a necessidade de comparar as respostas dos usuários ao utilizar também o `Protégé`. Os trabalhos relacionados vistos no Capítulo 5 também são ferramentas visuais, porém comparadas ao `Medina`, não possuem todas as funcionalidades e consequentemente não seria possível verificar respostas dos usuários em relação a visualização de inferências com a expressividade do `Medina`, por exemplo.

Após a execução do Experimento 1, o `Medina` foi atualizado com a implementação de novas funcionalidades. Foi visto a necessidade da adição do painel *Tree View* (Vide Seção 3.1.5.3), possibilitando uma navegação semelhante aos sistemas operacionais e o `Protégé`, por exemplo. Além disso, focando em uma modelagem menos confusa ao lidar com uma grande quantidade de elementos visuais, foram desenvolvidas as funcionalidades de ocultar axiomas e demais elementos visuais (Vide Seção 3.2.6).

Através dos questionários no Experimento 2 foi possível perceber uma preferência dos usuários participantes do experimento pela ferramenta `Medina` em todas as perguntas e

afirmações feitas. Além disso, através da análise qualitativa foi possível entender que a modelagem visual, associada ao uso de deduções de maneira automática foi bem visto pelos participantes do experimento.

Ao longo do Experimento 2 a ferramenta `Protégé` apresentou travamentos ao inserir classes com um nome relativamente grande (*HealthSurveillanceNotificationAction*, por exemplo). O observador sugeriu em todos os travamentos a reinicialização do `Protégé` e que os usuários a partir de então persistam a ontologia em disco a cada inserção de axiomas. Em alguns casos, porém, foram perdidos axiomas desenvolvidos pelos usuários até o momento do erro.

Mesmo com problemas apresentados durante o experimento, a ferramenta `Protégé` teve um tempo de modelagem menor que a ferramenta `Medina`. Através da análise qualitativa em ambos os experimentos, foi possível perceber que a modelagem visual acarreta em uma forma diferenciada de desenvolver ontologias, uma vez que os usuários sempre organizavam as entidades visuais, conferindo sua corretude e consequentemente modelando a ontologia em mais tempo em relação a abordagens de construção de axiomas estritamente textual como o `Protégé`. Além disso, a quantidade consideravelmente maior de nomes abreviados ao modelar a ferramenta `Protégé` no Experimento 2, influencia positivamente em seu tempo de desenvolvimento uma vez que com nomes menores é possível construir um axioma mais rápido e isso pode ter influenciado nos tempos de desenvolvimento.

Através da análise qualitativa das ontologias desenvolvidas no `Medina` foi possível perceber que o sistema proposto é capaz de representar visualmente ontologias, construir ontologias com expressividade e que também é capaz de realizar raciocínios de inferência e inconsistência.

Por ter uma percentagem de corretude em média, próximas, não é possível afirmar que a modelagem na ferramenta `Medina` influencia positivamente na corretude da modelagem dos axiomas desenvolvidos em relação a ferramenta `Protégé`. Por outro lado, o percentual de corretude próximo ao percentual do `Protégé` mostra também que utilizar a ferramenta proposta não acrescentou novos erros na modelagem de ontologias.

Com os resultados obtidos (questionários, ontologias desenvolvidas, corretude e tempo de modelagem) e através das análises quantitativa e qualitativa vistas neste capítulo, foi possível perceber que uma ferramenta de modelagem visual e dotada de raciocínio automático é útil para o aprendizado, construção e manutenção de ontologias expressivas. Dessa forma, os experimentos fornecem evidências de que a Hipótese de Pesquisa - H_1: *o* `Medina` ***provê*** *capacidade, corretude e facilidade na construção de ontologias expressivas e o* `Medina` ***provê*** *também capacidade de raciocínio de inferência e inconsistência a partir de modelagem visual* é verdadeira.

4.4 CONCLUSÕES DO CAPÍTULO

Durante a realização dos experimentos, obteve-se resultados que evidenciam que o `Medina` é capaz de:

- Construir ontologias em OWL DL 2 com expressividade de alto nível (máxima $\mathcal{ALCHIQ}$) a partir de modelagem visual;
- Realizar raciocínio de subsunção deduzindo que classes são subclasses de outras, a partir de suas respectivas descrições;
- Detectar e verificar inconsistências em tempo de desenvolvimento nas ontologias construídas;
- Apresentar *visual explanations* das deduções obtidas em tempo de desenvolvimento;
- Mitigar dificuldades inerentes a construção manual e não visual de ontologias;

As capacidades do `Medina` foram verificadas ao longo do capítulo através dos diversos resultados obtidos, proporcionando evidências de que a hipótese de pesquisa é verdadeira, respondendo de forma satisfatória a questão de pesquisa definida:

- *De que maneira uma ferramenta de autoria inteligente para modelagem de ontologias de forma visual e que realiza raciocínio automático pode ser útil para o aprendizado, construção e manutenção de bases de conhecimento modeladas como ontologias?*

Dessa forma, o estado da arte foi contemplado com uma nova solução, viável para o problema descrito. Novos problemas foram percebidos a partir do desenvolvimento do `Medina`, sendo necessário seu estudo por parte de outros pesquisadores (Vide Capítulo 6). A seguir apresentamos os trabalhos relacionados.

5 TRABALHOS RELACIONADOS

O desenvolvimento do sistema proposto foi baseado em trabalhos existentes na literatura. Trabalhos voltados ao desenvolvimento de ferramentas de criação e edição de ontologias de maneira visual foram estudados. Esses trabalhos possuem diferentes focos e níveis de detalhamento em relação ao `Medina`. A seguir são apresentados os trabalhos relacionados a ferramenta proposta.

5.1 AVONED

`AVOnEd` - *Aspect-Oriented Visual Ontology Editor* (HALLAY et al., 2017; GESTERKAMP; REBSTADT; MERTENS, 2017) é uma ferramenta com proposta de edição orientada a aspectos, sendo possível visualizar e editar um subconjunto de uma ontologia, mas que possui elementos semanticamente conectados. A orientação a aspectos é a separação em visões das mesmas entidades. Segundo Rebstadt et al. (2016), um exemplo da visão em aspectos é a personagem *Clark Kent*. Em uma perspectiva ele é um jornalista do planeta diário e na outra é o *Superman*, possuindo poderes, capaz de voar, etc. Mesmo sendo expectativas diferentes, as informações apresentadas são do mesmo indivíduo.

Os autores da ferramenta `AVOnEd` acreditam que utilizando essa separação das entidades de acordo com visões de aspecto é possível reduzir a complexidade da modelagem de ontologias, ter uma nova perspectiva e poder acrescentar mais conhecimentos a base. A ferramenta limita-se a utilizar construtores de interseção e quantificador existencial e possui checagem de consistência em tempo de execução sem uso de raciocinadores externos. Por utilizar a visão de aspectos no *software*, a visualização e edição de ontologias é feita apenas em partes da ontologia, separadas por mapas de aspectos. Modelando com os mapas de aspectos, de forma implícita, já é apresentado aos usuários partes da ontologia, mostrando apenas os elementos que possuem alguma relação com as entidades apresentadas no mapa de aspectos.

Na Figura 49 é possível visualizar a tela principal da ferramenta `AVOnEd`. A figura foi obtida de Hallay et al. (2017), uma vez que não foi possível utilizar a ferramenta. Na área 3 da Figura 49 é possível visualizar dois mapas de aspectos. No primeiro (mais à esquerda da área 3 da Figura 49) é apresentada uma visão relacionada ao planeta diário e relacionamentos como *Clark Kent* tem como profissão jornalista. No segundo mapa (mais à direita da área 3 da Figura 49) é apresentada a visão do super-herói *Superman* e um dos relacionamentos apresentados é que o *Superman* tem o poder de voar.

A representação visual da ferramenta (REBSTADT et al., 2016) é um grafo onde as classes, indivíduos e relações são os vértices, enquanto a ligação das relações com as classes ou indivíduos é feita utilizando arcos tracejados. A hierarquia de classes é apresentada

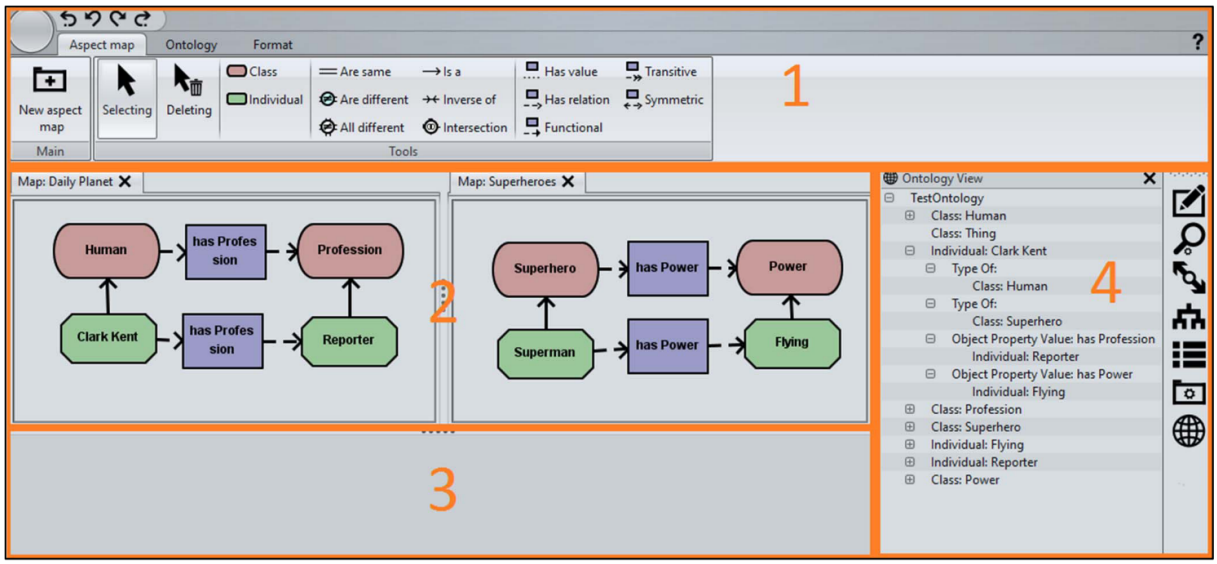


Figura 49 – Tela Principal da ferramenta AVOnEd. Fonte: Hallay et al. (2017)

através de arcos completos (não tracejados). Do ponto de vista dos construtores visuais da linguagem, existe uma sobrecarga dos mesmos. Na área 1 da Figura 49 é possível ver o construtor *Is a*, ele é utilizado para representar hierarquia de classes, propriedades e asserção de classes. Na ferramenta Medina a sobrecarga de construtores é evitada, uma vez que representar relacionamentos diferentes com o mesmo construtor visual pode prejudicar a legibilidade do modelo.

Tratando-se do raciocínio, a linguagem visual não apresenta funcionalidades visuais que representem inferências, uma característica relevante quando são desenvolvidas ontologias em OWL 2 e implementada pela ferramenta Medina. No *software* AVOnEd o usuário não pode inserir axiomas que causem inconsistência na ontologia. O processo de raciocínio da ferramenta foi desenvolvido internamente, sem o uso de raciocinadores externos. No Medina são utilizados os raciocinadores Pellet (PARSIA; SIRIN, 2004) e HermiT (GLIMM et al., 2014), já utilizados e consolidados pela comunidade.

Conforme pode ser visto na Tabela 9, os construtores de união, complemento, e quantificador universal não são representados visualmente na ferramenta AVOnEd, mas foram implementados na ferramenta Medina. Quanto ao processo de raciocínio, ambas as ferramentas implementam a checagem de consistência, sendo Medina o único entre os dois *softwares* a implementar a funcionalidade de inferência.

As propriedades transitivas, funcionais e simétricas são implementadas de maneira visual na ferramenta AVOnEd, diferentemente da ferramenta Medina que não implementa tais funcionalidades. A ferramenta proposta, até o presente momento, teve seu desenvolvimento focado na capacidade de apresentar *feedback* aos usuários a respeito do que está sendo modelado. As funcionalidades relacionadas as propriedades estão previstas para serem implementadas na ferramenta proposta em uma nova versão.

Já em relação as técnicas de visualização das ferramentas, AVOnEd e Medina usam as

Tabela 9 – Comparação de funcionalidades entre as ferramentas AVOnEd e Medina

Funcionalidade	AVOnEd	Medina
União (Visual)	Não	Sim
Interseção (Visual)	Sim	Sim
Complemento (Visual)	Não	Sim
Asserção (Visual)	Sim	Sim
Quantificador Existencial (Visual)	Sim	Sim
Quantificador Universal (Visual)	Não	Sim
Propriedades Transitivas	Sim	Não
Propriedades Funcionais	Sim	Não
Propriedades Simétricas	Sim	Não
Consistência	Sim	Sim
Inferência	Não	Sim
Node-link and Tree	Sim	Sim
Idented List	Sim	Sim
Visualização completa da ontologia	Não	Sim

mesmas técnicas: *Node-link and Tree* (ver Seção 2.4 e *Idented List*, técnicas que visualizam ontologias como um grafo direcionado e uma árvore, respectivamente. Na visualização da ontologia de maneira completa, a ferramenta AVOnEd não possui tal funcionalidade enquanto a ferramenta proposta é capaz de representar visualmente toda a ontologia.

5.2 EDDY

Eddy (LEMBO et al., 2016), é considerado por seus autores como o primeiro editor de ontologias que permite criar ontologias em OWL 2 ($\mathcal{SROIQ}(D)$) usando apenas funcionalidades gráficas. Utiliza a linguagem Graphol (CONSOLE et al., 2014), onde conceitos, relacionamentos e atributos são representados como vértices de um grafo.

Na Figura 50 é possível visualizar a tela principal da ferramenta Eddy. Ao centro da tela os usuários realizam a modelagem visual de suas ontologias. Do lado esquerdo é possível escolher quais construtores visuais serão utilizados na modelagem e ao lado direito é possível visualizar informações sobre a ontologia modelada.

Expressões complexas em OWL 2 são possíveis de modelar utilizando operadores gráficos, tais como união, interseção, restrições. As restrições universal e existencial são conectadas através de arcos a um relacionamento e a um conceito ou um operador gráfico. Esse tipo de representação visual no Eddy necessita que o usuário conecte as três entidades que fazem parte do relacionamento: a propriedade, o domínio e a imagem. Entretanto, a adição de novas restrições acarretam em uma sobreposição de arcos que, a depender do tamanho da ontologia, pode prejudicar a visualização. Considere o exemplo mostrado na

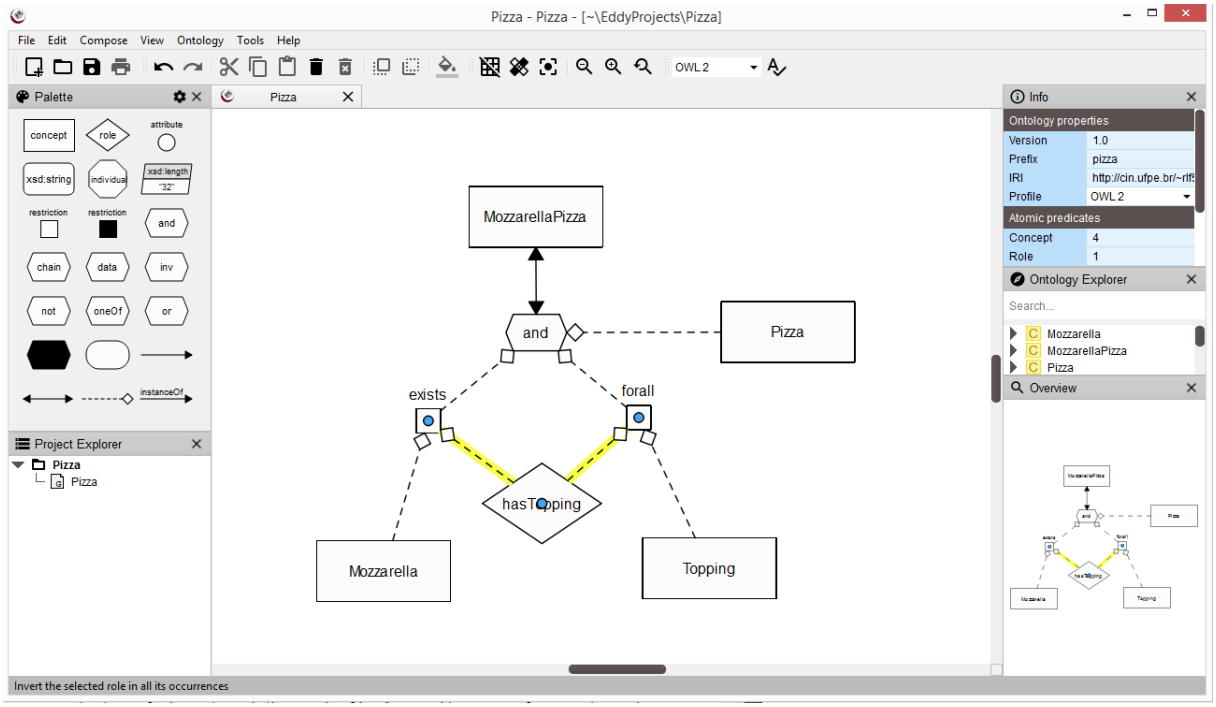


Figura 50 – Tela Principal da ferramenta Eddy.

Figura 50.

Na Figura 50 é possível visualizar duas instâncias da propriedade *hasTopping*, uma utilizando o quantificador existencial e outra o quantificador universal. Todas as duas instâncias possuem uma ligação com a entidade visual que representa a propriedade *hasTopping*. Já na ferramenta Medina, o mesmo fato é modelado sem a ligação através de arcos com a entidade visual que representa a propriedade *hasTopping* (ligações destacadas na Figura 50. Ao invés da ligação o nome da propriedade é utilizado na própria instância do quantificador existencial e universal.

Se tratando de funcionalidades relacionadas ao raciocínio, a ferramenta não executa nenhuma tarefa como checagem de consistência ou inferências e a linguagem visual Graphol não possui construtores visuais também relacionados ao raciocínio. O processo de raciocínio auxilia os usuários na criação de ontologias consistentes. Além disso, a ferramenta Eddy permite a construção de representações visuais que não podem ser geradas em código OWL, o que pode prejudicar o desenvolvimento por parte de usuários iniciantes. Diferente do Eddy, a ferramenta proposta possui funcionalidades relacionadas ao raciocínio tais como checagem de inconsistência e inferência de subsunções.

A ferramenta, até a data em que foi realizado o download[1], não possuía capacidade de importar uma ontologia em OWL 2 ou visualizar o código OWL internamente, o que é útil para quem estiver modelando poder entender como a representação visual está sendo codificada. A ferramenta Medina, diferentemente do Eddy, possibilita aos usuários importar ontologias e visualizar em tempo real o código OWL 2 gerado.

[1] http://www.dis.uniroma1.it/graphol/download.html. Acessado em 26/01/2019.

Tabela 10 – Comparação de funcionalidades entre as ferramentas `Eddy` e `Medina`

Funcionalidade	`Eddy`	`Medina`
Expressividade DL	$\mathcal{SROIQ}(\mathcal{D})$	$\mathcal{ALCHIQ}$
Importa ontologias em OWL	Não	Sim
Visualizar código OWL	Não	Sim
Raciocínio Automático	Não	Sim
Node-link and Tree	Sim	Sim
Idented Tree	Sim	Sim
Ocultar entidades visuais	Não	Sim

Na Tabela 10 é possível visualizar a comparação entre algumas das funcionalidades presentes nas ferramentas `Eddy` e `Medina`. A ferramenta `Eddy` possui uma expressividade maior na construção de ontologias que o `Medina`. Já em relação a funcionalidades como importar ontologias em OWL, visualizar o código OWL ou raciocínio automático não são suportados pela ferramenta `Eddy`. A ferramenta `Medina` implementa essas funcionalidades. Importar outras ontologias OWL já existentes é uma tarefa crucial para ferramentas de edição de ontologias uma vez que o formato OWL é o recomendado pela W3C.

Tratando-se das técnicas de visualização, ambas as ferramentas implementam as técnicas *Node-link and Tree* e *Idented Tree*. A ferramenta `Eddy`, todavia, não possui funcionalidades para ocultar componentes visuais como classes ou propriedades. Já a ferramenta proposta possui funcionalidades para ocultar entidades visuais como classes, propriedades e indivíduos além de permitir a ocultação de axiomas complexos.

5.3 GROWL

É uma ferramenta de edição em que utiliza cores, tamanhos e sombras como uma forma de codificar propriedades básicas da linguagem OWL (KRIVOV; WILLIAMS; VILLA, 2007). Além disso, a ferramenta utiliza vértices no grafo para representar classes, propriedades e também axiomas. O que pode ser uma desvantagem, visto que a quantidade de vértices gerados pela ferramenta será bem maior do que existir somente um arco representando o axioma, como a ferramenta `Medina`.

Na Figura 51 é possível visualizar a tela principal da ferramenta `GrOWL`, adicionada de Krivov, Williams e Villa (2007), uma vez que não foi possível utilizá-la. Na esquerda da Figura 51 é possível visualizar uma paleta para adição de construtores visuais na modelagem e também um painel utilizando a técnica de visualização *Idented Tree* para representar a hierarquia de classes participantes da modelagem. Do lado direito da Figura 51 é possível visualizar um formulário de alteração de informações dos elementos visuais. No caso da Figura 51 o usuário está selecionando o nome da classe *Camera*.

No centro da tela inicial da ferramenta `GrOWL` está presente a ontologia, representada

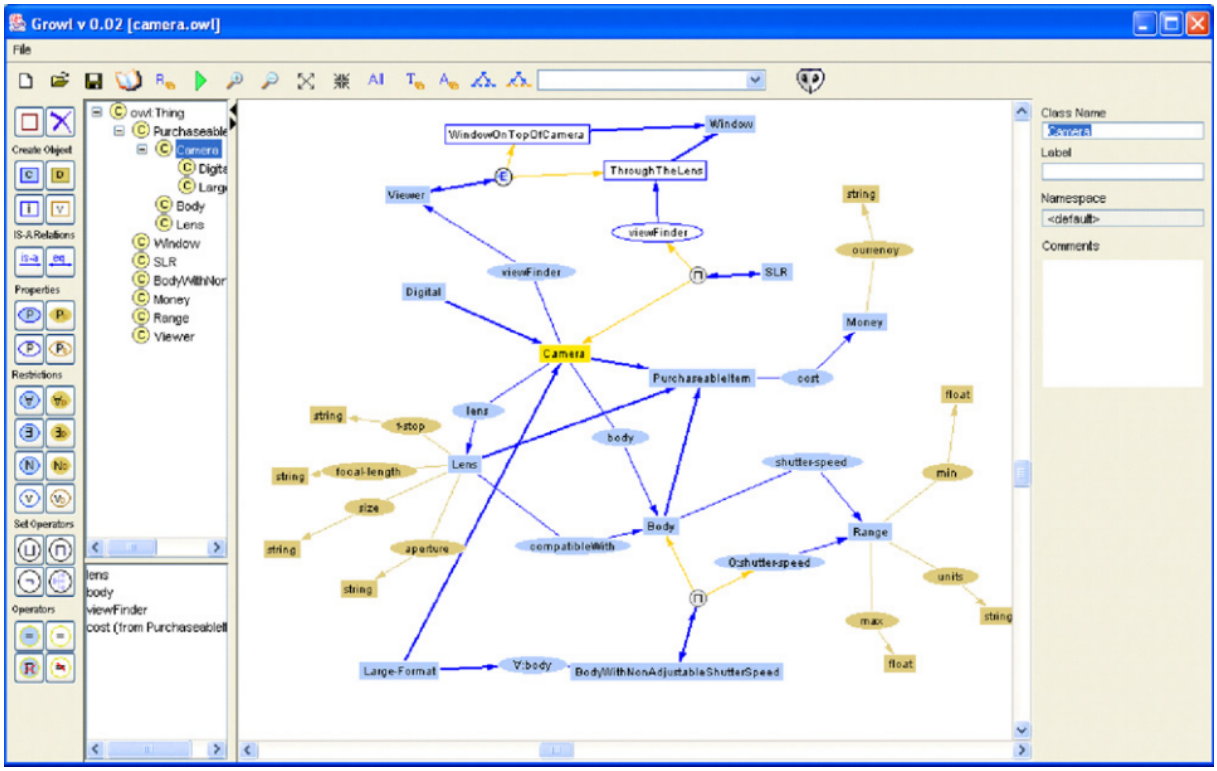


Figura 51 – Tela Principal da ferramenta GrOWL. Fonte: Krivov, Williams e Villa (2007)

como um grafo. Construtores de expressões de classe, tais como união, interseção, e os quantificadores estão presentes na ferramenta. Entretanto sua representação visual é através de seus símbolos em DL. A ferramenta proposta representa os construtores através de uma descrição textual, baseada na *Manchester Syntax*.

Na ferramenta GrOWL foram implementadas também técnicas de visualização de acordo com o *feedback* dos usuários da ferramenta, filtros para que só sejam vistas definições de classes, hierarquia e instâncias associadas a um nó específico selecionado e busca de nós utilizando uma busca incremental foram utilizados.

Segundo Krivov, Williams e Villa (2007) a combinação de filtro e técnicas de navegação, proposta pela experiência dos usuários do GrOWL, são técnicas de visualização úteis e que a ferramenta apresenta resultados interessantes ao aplicar esses filtros em ontologias grandes uma vez que a quantidade de nós e arcos apresentados visualmente diminui. A ferramenta Medina possui entre suas funcionalidades a opção de ocultar componentes visuais, tais como classes, propriedades, indivíduos e construtores de classe. É possível ocultar elementos específicos ou até ocultar todos os tipos de uma só vez, ocultar todas as propriedades, por exemplo.

A ferramenta não possui funcionalidades relacionadas a raciocínio automático, tais como verificação de inconsistências e deduções e, de acordo com o artigo publicado (KRIVOV; WILLIAMS; VILLA, 2007) descrevendo a ferramenta, a mesma não utiliza o código OWL mais recente (OWL 2). A ferramenta Medina, nestas funcionalidades, tem vantagem em relação ao GrOWL por ter como referência o código OWL 2, versão mais recente,

e possuir funcionalidades relacionadas ao raciocínio.

Tabela 11 – Comparação de funcionalidades entre as ferramentas GrOWL e Medina

Funcionalidade	GrOWL	Medina
Expressividade DL	$\mathcal{SHOIN}(\mathcal{D})$	$\mathcal{ALCHIQ}$
Versão da OWL suportada	1	2
Raciocínio Automático	Não	Sim
Node-link and Tree	Sim	Sim
Idented Tree	Sim	Sim
Ocultar entidades visuais	Sim	Sim

A expressividade da ferramenta GrOWL é superior a ferramenta proposta, conforme pode ser visto na Tabela 11. Entretanto, a versão da OWL suportada pela ferramenta GrOWL é a OWL 1 enquanto a ferramenta Medina suporta a versão mais recente, OWL 2. O raciocínio automático é uma desvantagem da ferramenta GrOWL em relação ao Medina, uma vez que GrOWL não a implementa.

As técnicas visuais *Node-link and Tree* e *Idented Tree* são implementadas por ambas as ferramentas, incluindo também as opções de ocultar os componentes visuais presentes na ontologia.

5.4 ONTOTRACK

Com a ferramenta OntoTrack (LIEBIG; NOPPENS, 2005) é possível editar ontologias de forma visual, onde cada alteração feita é sincronizada com um raciocinador retornando graficamente as consequências desta alteração. Classes insatisfatíveis são representadas com bordas na cor vermelha.

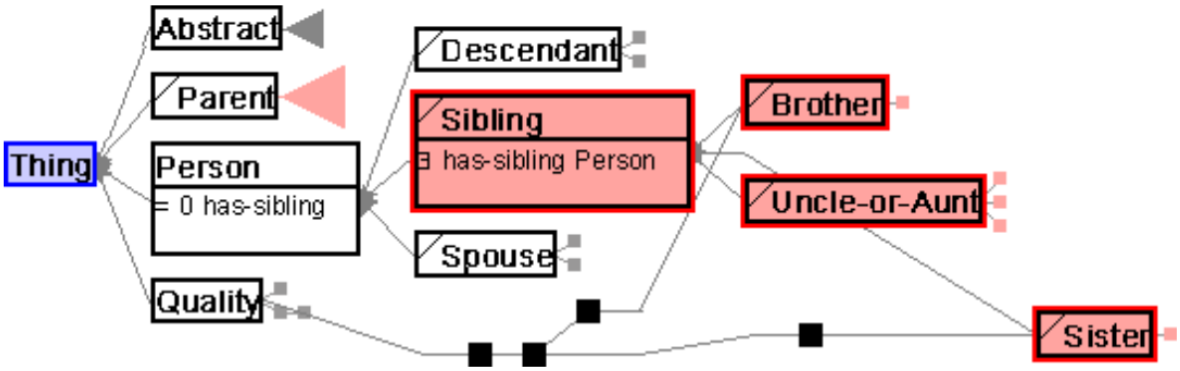


Figura 52 – Tela Principal da ferramenta OntoTrack. Fonte: Liebig e Noppens (2005)

Na Figura 52 é possível visualizar a checagem de consistência da ferramenta OntoTrack. Através de raciocínio automático, é apresentado ao usuário que a classe *Sibling* é insatisfatível. A classe *Sibling* tem como restrição ter pelo menos um relacionamento *has-sibling*

com um indivíduo da classe *Person*. Entretanto, de acordo com a modelagem *Sibling* é subclasse de *Person* que possui uma restrição de cardinalidade indicando que não pode ter uma relação *has-sibling*, gerando assim a insatisfatibilidade. Consequentemente, todas as subclasses de *Sibling* também são identificadas como insatisfatíveis.

Nota-se na Figura 52 que para a construção de axiomas que utilizem restrições, seu desenvolvimento deve ser interno a classe em questão. Essa notação assemelha-se a notação UML. Porém levando em consideração que as restrições também são classes, essa abordagem possui como principal desvantagem a representação visual de classes ora como componentes visuais (classes nomeadas), ora como restrições (restrições). Na ferramenta `Medina` a representação visual tanto de classes nomeadas como de expressões de classe é na forma de componentes visuais e estes consequentemente conectados traduzindo assim o significado de representação e hierarquia de classes.

O `OntoTrack` suporta um subconjunto raciocinável da OWL *Lite* denominado como OWL $Lite^{-}$. Neste subconjunto, além da exclusão dos construtores para união e complementar oriundos da OWL *Lite*, é permitido apenas uma definição para as classes. Seja uma classe C, no `OntoTrack` só é permitida a inserção de axiomas de equivalência ou de subclasse com a classe C a esquerda. Além disso, a classe C deverá ser nomeada, ou seja, atômica. O axioma $A \sqcap B \sqsubseteq \exists r.D$, por exemplo, não pode ser modelado assim na ferramenta. Na ferramenta `Medina` podem ser adicionados tantos axiomas de definição quanto desejados pelos usuários, sem necessidade de classe nomeada do lado esquerdo dos axiomas. Segundo Liebig e Noppens (2005) essas restrições foram implementadas por acreditarem que foi originalmente pretendido pelos próprios designers da linguagem OWL. No entanto, esse tipo de restrição é uma desvantagem da ferramenta `OntoTrack` já que axiomas modelados em DL da maneira não permitida, devem ser adaptados, quando possível.

Para mostrar as restrições relacionadas a propriedades, o `OntoTrack` utiliza um conceito semelhante ao diagrama de classes do UML (*Unified Modeling Language*), onde as restrições específicas de uma classe são inseridas na posição inferior da mesma, utilizando uma sintaxe DL. As restrições suportadas são as restrições de cardinalidade, além das restrições dos quantificadores existencial ($\exists$) e universal ($\forall$).

Além das funcionalidades de raciocínio subsunção e insatisfatibilidade, o `OntoTrack` apresenta o conjunto de axiomas que causam as inferências obtidas de maneira textual, utilizando um algoritmo semelhante ao *Tableaux* (LIEBIG; HALFMANN, 2005) – técnica de raciocínio utilizada em diversos motores de raciocínio como `Pellet` (PARSIA; SIRIN, 2004) e `HermiT` (GLIMM et al., 2014) – sendo uma das poucas ferramentas de modelagem visual a implementar essas funcionalidades.

Em uma comparação entre as ferramentas `OntoTrack` e `Medina` verifica-se que a ferramenta `Medina` é mais expressiva que o `OntoTrack`, a comparação entre as ferramentas pode ser vista na Tabela 12. Sobre a funcionalidade de raciocínio automático, ambas as ferra-

Tabela 12 – Comparação de funcionalidades entre as ferramentas `OntoTrack` e `Medina`

Funcionalidade	`OntoTrack`	`Medina`
Expressividade DL	$Lite^{-}$	$\mathcal{ALCHIQ}$
Raciocínio Automático	Sim	Sim
Raciocinadores utilizados	`RACER`	`Pellet` e `Raccoon`
Apresentação dos axiomas causadores das inferências	Textual	Textual e Visual
Quantidade de definições por classe	1	Ilimitado
Classes não atômicas no lado esquerdo dos axiomas	Não	Sim
Node-link and Tree	Sim	Sim
Idented Tree	Não	Sim
Ocultar entidades visuais	Sim	Sim

mentas implementam formas de o usuário obter inferência sobre a ontologia modelada. Quanto a apresentação dos axiomas que causam os raciocínios a ferramenta `OntoTrack` apresenta de maneira textual enquanto a ferramenta proposta possui apresenta tanto de forma textual quanto visual (*visual explanation*), destacando os arcos que conectam os axiomas que fazem parte do conjunto de axiomas utilizado.

O uso de raciocinadores externos é feito pelas duas ferramentas comparadas. O `OntoTrack` utiliza o motor de raciocínio `RACER` (HAARSLEV; MÖLLER, 2001), enquanto a ferramenta proposta possibilita o usuário a utilizar dois motores de raciocínio: Pellet (PARSIA; SIRIN, 2004) e Raccoon (FILHO; FREITAS; OTTEN, 2017). O total de definições para uma classe específica é limitado para o `OntoTrack` e ilimitado para o `Medina`. Além disso, o `OntoTrack` não permite o uso de classes não atômicas à esquerda dos axiomas, podendo apenas ser utilizadas classes nomeadas (atômicas). A ferramenta proposta possibilita ao usuário inserir axiomas com classes não atômicas em ambos os lados dos axiomas.

A ferramenta `OntoTrack` tem como desvantagem visual a não utilização da técnica *Idented Tree*, implementada pelas ferramentas `Medina`, `GrOWL`, `Eddy` e `AVOnEd`. Essa técnica é uma forma eficiente e intuitiva para os usuários navegarem pela hierarquia de classes da ontologia a ser desenvolvida (KATIFORI et al., 2007). A técnica de visualização *Node-link and Tree* é utilizada por ambas as ferramentas. A ocultação de entidades visuais é feita pelo `OntoTrack` através de novos símbolos, na Figura 52 as subclasses da classe *Parent*, por exemplo, foram omitidos da visualização.

5.5 OWLGRED

A ferramenta `OWLGrEd` (BĀRZDIŅŠ et al., 2010) é um editor gráfico OWL baseado nos diagramas de classe do UML. Como nem todos os conceitos de OWL possuem conceitos equivalentes nos diagramas de classe UML, o `OWLGrEd` amplia os conceitos dos diagramas de classe UML adicionando a sintaxe de Manchester para os conceitos não implementados

visualmente do OWL.

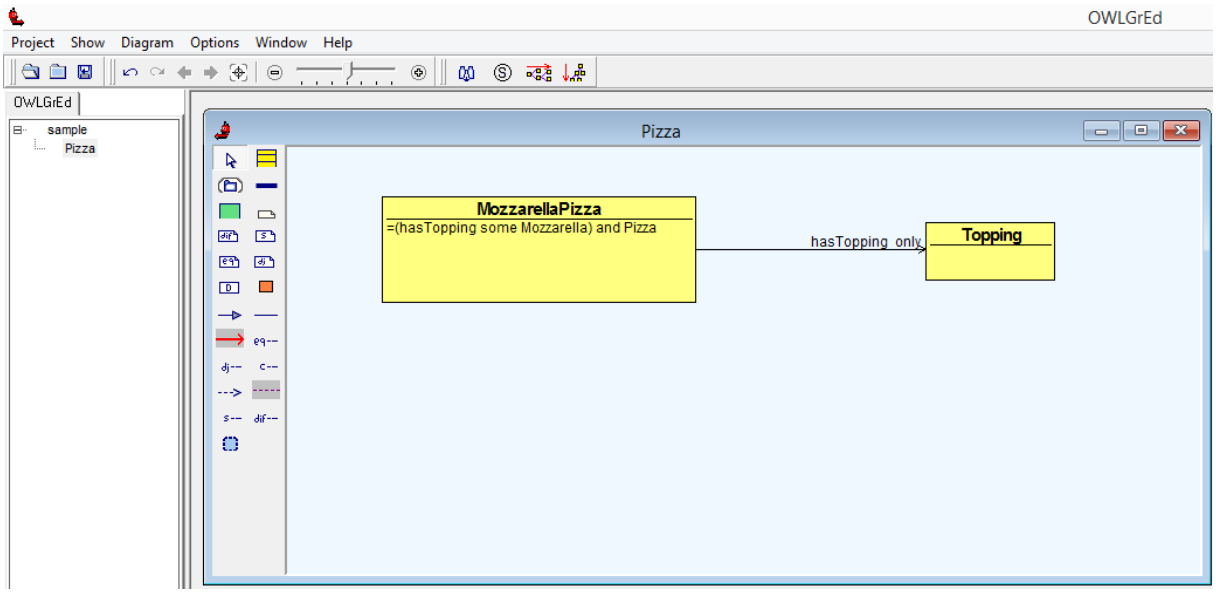


Figura 53 – Tela Principal da ferramenta `OWLGrEd`.

Na Figura 53 é possível ver a modelagem de dois axiomas na ferramenta `OWLGrEd`. A adição de axiomas com classes complexas pode ser feita de duas formas: visualmente e textualmente. Entretanto, a modelagem visual da ferramenta não cobre todas as representações textuais.

Uma das principais desvantagens do `OWLGrEd` em relação a ferramenta proposta pode ser vista nessa figura. A classe *MozzarellaPizza*, possui um axioma de equivalência, representado internamente em notação textual, semelhante a ferramenta `OntoTrack`. Já um axioma de subclasse envolvendo a classe *MozzarellaPizza* e o quantificador universal ($\forall$) foi realizado de maneira visual. Axiomas com os construtores de união ($\sqcup$), interseção ($\sqcap$) e complemento ($\neg$) não são possíveis de representar visualmente pelo `OWLGrEd`. Na ferramenta `Medina`, todos os construtores disponíveis na ferramenta podem ser construídos de maneira visual e também de maneira textual, através da adição de axiomas em *Manchester Syntax*.

Um dos serviços oferecidos pelo `OWLGrEd` é a refatoração que altera a notação gráfica sem alterar a semântica dos conceitos, permitindo ao usuário escolher o estilo gráfico que mais se adeque a sua necessidade. Por exemplo, escolher se uma asserção entre uma propriedade e duas classes será representada como uma associação entre as classes ou como uma propriedade da representação UML, inserida dentro de uma das classes relacionadas.

Ter essa variedade de notação permite representar melhor grandes ontologias pois é possível diminuir a quantidade de arcos ligando as entidades, utilizando a notação de propriedade do UML (BĀRZDIŅŠ et al., 2010). No entanto, nem todos os construtores textuais possuem versões visuais e consequentemente a refatoração para a representação visual nem sempre será completa.

Uma funcionalidade disponível no OWLGrEd é a visualização e edição de fragmentos da ontologia (LIEPINŠ; CERANS; SPROGIS, 2012), baseada na decomposição automática. Um usuário seleciona algumas entidades que devem aparecer no fragmento e então o fragmento é ampliado com todos os axiomas relevantes para estas entidades. Existe também uma versão do OWLGrEd para visualização de ontologias *online* (LIEPINŠ; GRASMANIS; BOJARS, 2014) não sendo possível ainda editar ontologias. A ferramenta Medina não possibilita o usuário editar apenas fragmentos da ontologia, mas possibilita a opção de ocultar elementos visuais não necessários, o que na prática tem resultado parecido à abordagem do OWLGrEd.

Usuários que não estão acostumados a utilizar a linguagem UML podem sentir dificuldade para compreender e desenvolver ontologias com esta ferramenta. Além disso, o uso de texto em sintaxe de Manchester para o desenvolvimento de axiomas complexos diminui a quantidade de conexões visuais, mas pode ser confuso para o usuário, uma vez que as classes participantes aos axiomas não são relacionadas de forma visual. Na Figura 53 a classe Pizza existe devido ao axioma de equivalência presente em $MozzarellaTopping$, porém não é representado visualmente.

Por fim, a ferramenta OWLGrEd possui integração com o Protégé, mas sua linguagem visual ou sua versão *Desktop* não possuem funcionalidades relacionadas ao raciocínio.

Tabela 13 – Comparação de funcionalidades entre as ferramentas OWLGrEd e Medina

Funcionalidade	OWLGrEd	Medina
União (Visual)	Não	Sim
Interseção (Visual)	Não	Sim
Complemento (Visual)	Não	Sim
Asserção (Visual)	Sim	Sim
Raciocínio Automático	Não	Sim
Node-link and Tree	Sim	Sim
Idented Tree	Não	Sim
Edição de Fragmentos	Sim	Sim*

Na Tabela 13 é possível visualizar a comparação de algumas funcionalidades da ferramenta OWLGrEd e a ferramenta proposta. Os construtores de união, interseção e complemento não são implementados visualmente pelo OWLGrEd, sendo essa uma desvantagem em relação ao Medina. A asserção de indivíduos é realizada por ambas as ferramentas. O raciocínio automático, funcionalidade importante na construção de uma ontologia, é implementada apenas pela ferramenta proposta.

Quanto a visualização das ontologias, OWLGrEd e Medina implementam a mesma técnica: *Node-link and Tree*. A técnica de visualização *Idented Tree*, no entanto, é suportada apenas pelo Medina. Além disso, em relação a edição de partes da ontologia, o OWLGrEd implementa tal funcionalidade. Levando-se em consideração a capacidade do Medina de

ocultar elementos visuais, a funcionalidade de edição de fragmentos também é implementada pela ferramenta proposta.

5.6 OWLAX

OWLAx (SARKER; KRISNADHI; HITZLER, 2016) é um *plugin* para o Protégé que possibilita a criação de axiomas visualmente. Com o OWLAx é possível construir uma representação gráfica de classes e a partir dessas representações, construir axiomas. De acordo com Sarker, Krisnadhi e Hitzler (2016), o OWLAx não é para visualizar uma ontologia, mas sim para facilitar a criação de diagramas de classes gráficas dentro do Protégé e fornecer uma maneira de gerar axiomas a partir dele, eliminando a necessidade de usar ferramentas separadas para criar o diagrama de classes.

Em relação as funcionalidades do OWLAx, foram percebidas duas principais desvantagens em relação a ferramenta proposta: limitação de expressividade e necessidade de escolha dos axiomas que são gerados. Por não permitir o uso de construtores como interseção ($\sqcap$), união ($\sqcup$) e complemento ($\neg$) o *plugin* restringe a expressividade de axiomas que podem ser construídos com sua notação visual. Além disso, ao desenvolver um diagrama visualmente é necessário escolher qual tipo de axioma deverá ser integrado ao Protégé.

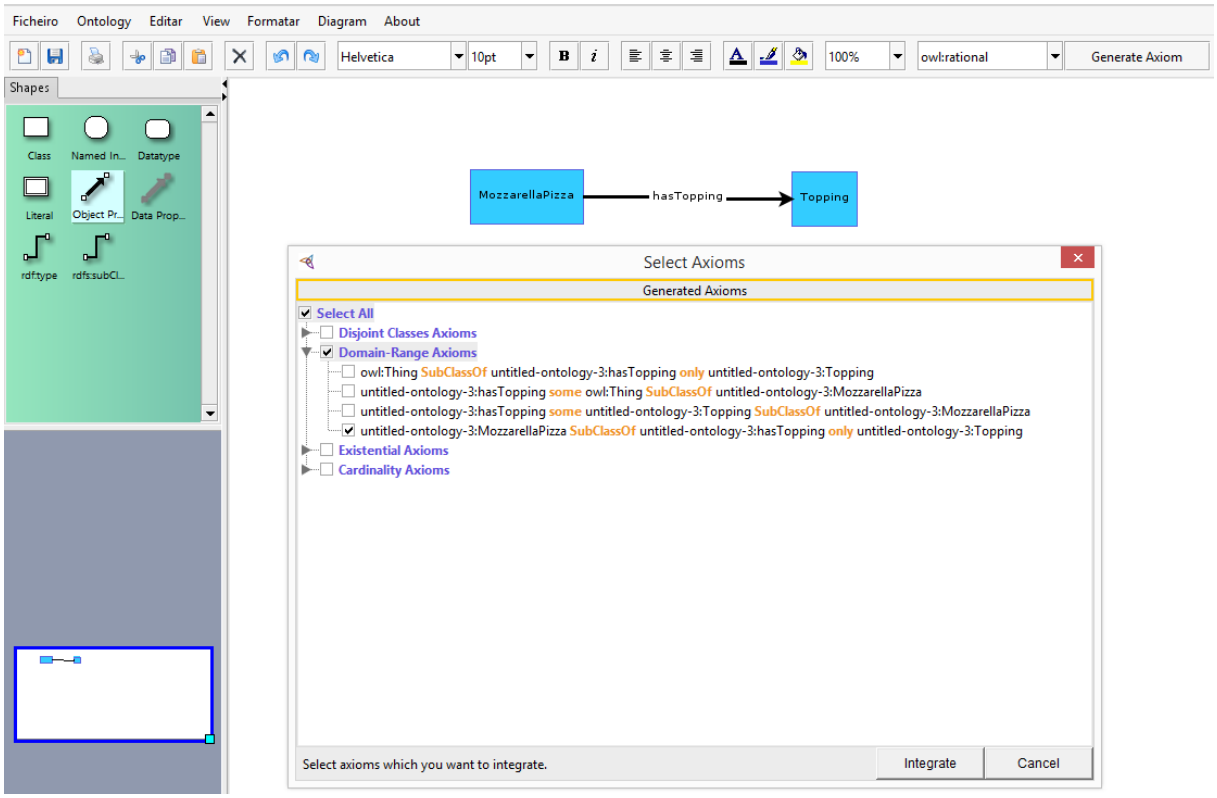


Figura 54 – Tela Principal do *plugin* OWLAx.

Na Figura 54 é possível visualizar a tela principal do *plugin* OWLAx. No momento registrado na captura de tela, foi desenvolvida uma relação, nomeada *hasTopping* entre as classes *MozzarellaPizza* e *Topping*. A intenção do usuário é desenvolver o axioma

$MozzarellaPizza \sqsubseteq \forall hasTopping.Topping$. Como o *plugin* não possui construções visuais explicitas para a criação do quantificador universal ($\forall$), o usuário deverá escolhe-lo entre uma variedade de opções. Na ferramenta proposta todos os quantificadores disponíveis e os construtores de união, interseção e complemento estão disponíveis para o usuário, permitindo uma maior expressividade na construção de axiomas, sem a necessidade de escolha a respeito de sua modelagem visual qual axioma deve ser gerado.

O raciocínio automático não é implementado pelo *plugin*, dado que o próprio Protégé tem funcionalidades a esse respeito. Entretanto, na modelagem visual também não é possível obter representações dos axiomas inferidos. Sendo necessário que o usuário visualize os axiomas deduzidos na aba de classes, de maneira textual. No `Medina` as inferências e insatisfatibilidades são apresentadas ao usuário visualmente adicionadas de *visual explanation*, destacando os axiomas que estão relacionados aquela inferência obtida.

O *plugin* `OWLAx` implementa somente uma técnica de visualização: *Node-link and Tree*. É justificável não implementar a técnica de visualização *Idented Tree* já que o Protégé a implementa. Entretanto, a visualização de ontologias não é realizada no `OWLAx`, sendo necessária a interoperação por parte do usuário entre o `OWLAx` e outros *plugins* do Protégé. O que é uma desvantagem com a ferramenta `Medina` uma vez que a visualização e edição de ontologias está disponível no mesmo local.

Tabela 14 – Comparação de funcionalidades entre as ferramentas `OWLAx` e `Medina`

Funcionalidade	`OWLAx`	`Medina`
União (Visual)	Não	Sim
Interseção (Visual)	Não	Sim
Complemento (Visual)	Não	Sim
Ambiguidade de construtores visuais	Sim	Não
Raciocínio Automático (Visual)	Não	Sim
Node-link and Tree	Sim	Sim
Idented Tree	Não	Sim
Visualização de ontologias	Não	Sim

Na Tabela 14 são mostradas algumas funcionalidades e um comparativo entre o `OWLAx` e o `Medina`. União, interseção e complemento não são implementados pelo *plugin* `OWLAx` e estão presentes na ferramenta `Medina`. No `OWLAx` representações visuais são ambíguas, sendo necessário que o usuário selecione qual axioma deve ser gerado. No `Medina` todos os construtores são implementados separadamente, evitando a ambiguidade e garantindo que o usuário obterá o axioma que foi modelado. Mesmo sendo um *plugin* do Protégé, o `OWLAx` não permite a visualização de novos axiomas obtidos através de raciocínio automático, diferentemente do `Medina`. A respeito da representação visual, o `OWLAx` utiliza apenas a técnica *Node-link and Tree* enquanto o `Medina` utiliza *Node-link and Tree* e *Idented*

Tree, além de permitir a visualização de ontologias. Na seção seguinte é apresentada a ferramenta proposta.

5.7 MEDINA - A FERRAMENTA PROPOSTA

`Medina` é uma ferramenta desenvolvida com foco na representação de conhecimento em DL. Sua expressividade atual é $\mathcal{ALCHIQ}$ e permite que o usuário desenvolva axiomas utilizando classes complexas. O código OWL é gerado em tempo de execução, a cada alteração visual da ontologia. No momento, `Medina` ainda não trabalha com as *data properties*, propriedades OWL 2 relacionadas aos tipos de dados (inteiro, booleano, *string*, etc.) e consequentemente sua expressividade atual ainda é inferior a ferramenta `Eddy`, por exemplo, que possui expressividade $\mathcal{SROIQ}(\mathcal{D})$.

A ferramenta possui funcionalidades relacionadas ao raciocínio como checagem de inconsistência e inferência de novos axiomas (raciocínio de subsunção) apresentadas de maneira visual automaticamente. Além de apresentar o raciocínio de maneira automática, `Medina` possui como inovação a apresentação do conjunto de axiomas que causam os raciocínios também de forma visual (*visual explanation*). Essa *visual explanation* do raciocínio permite que o usuário tenha *feedback* a respeito de sua modelagem ajudando o usuário a corrigir possíveis erros e também compreender a consequência lógica dos fatos modelados.

Na seção seguinte são mostrados todos os trabalhos relacionados e uma comparação entre todos esses trabalhos e a ferramenta proposta.

5.8 COMPARANDO AS FERRAMENTAS DE EDIÇÃO VISUAL DE ONTOLOGIAS

Para que possibilite uma melhor compreensão sobre as características de cada um dos trabalhos relacionados, desenvolvemos uma comparação levando em conta funcionalidades visuais das ferramentas citadas nos trabalhos.

Os critérios utilizados na comparação são o uso de construtores de classes união, interseção, complemento, raciocínio automático como checagem de inconsistência, inferência, axiomas de asserção, restrições (existencial, universal e de cardinalidade), axiomas de propriedade (relação inversa e subpropriedades) e o uso da *visual explanation*. Os resultados podem ser vistos na Tabela 15.

Tratando-se da utilização dos construtores de união e complemento somente as ferramentas `AVOnEd`, `OntoTrack` e o *plugin* `OWLAx` não possibilitam o uso desses construtores. Já o construtor de interseção é implementado por todas as ferramentas comparadas. As ferramentas `AVOnEd`, `OntoTrack` e `Medina` são as únicas a implementar a checagem de consistência, sendo a `OntoTrack` a única ferramenta (com exceção da ferramenta proposta) a implementar também a inferência de axiomas.

Tabela 15 – Tabela Comparativa das ferramentas de modelagem visual encontradas na revisão e a ferramenta proposta. Células com ✓indicam uma determinada característica que a ferramenta implementa (união, interseção, etc.). Células com Células com ✗indicam que a ferramenta não implementa a funcionalidade indicada. O `Medina` foi a única ferramenta, dentre as comparadas, que implementa todas as funcionalidades analisadas.

	AvOnEd	Eddy	OntoTrack	OWLGrEd	GrOWL	OWLAx	Medina
União	✗	✓	✗	✓	✓	✗	✓
Interseção	✓	✓	✓	✓	✓	✗	✓
Complemento	✗	✓	✗	✓	✓	✗	✓
Consistência	✓	✗	✓	✗	✗	✗	✓
Inferência	✗	✗	✓	✗	✗	✗	✓
Asserção	✓	✓	✗	✓	✓	✓	✓
Restrições	✓*	✓	✓	✓	✓	✓	✓
Propriedades	✓	✓	✓	✓*	✓	✗	✓
Visual Explanation	✗	✗	✗	✗	✗	✗	✓

Axiomas de asserção de conceitos e de relações são funcionalidades que não foram implementadas somente no `OntoTrack`. As restrições (existencial, universal e de cardinalidade) foram implementadas por todas as ferramentas, sendo a implementação da AvOnEd apenas pela barra lateral. Os axiomas de propriedade (inversa e subpropriedade) não foram implementados somente pelo *plugin* `OWLAx` e a implementação do `OWLGrEd` para essas propriedades é somente de maneira textual.

Em relação a *visual explanation*, `Medina` é a única ferramenta que realiza tal funcionalidade. A ferramenta `OntoTrack` apresenta o conjunto de axiomas que causam os raciocínios realizados utilizando como base um algoritmo de *Tableau*. A ferramenta `Medina` apresenta uma *visual explanation* de insatisfatibilidade das classes e deduções de axiomas, destacando os arcos – que são os axiomas visualmente – que causam aquele raciocínio. A ferramenta `Medina` é a única dentre as comparadas a implementar todas as funcionalidades em questão.

A seguir apresentamos a arquitetura do `Medina`, suas funcionalidades e tecnologias usadas em sua construção.

6 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho, foi apresentada a ferramenta Medina, uma ferramenta de autoria inteligente para construção de ontologias a partir de modelagem visual de componentes. O `Medina` fornece funcionalidades para o desenvolvimento de ontologias de forma simples e com raciocínio automático de subsunção e inconsistência.

Nosso objetivo principal foi o desenvolvimento de uma ferramenta de autoria inteligente para construção, manipulação e modificação de ontologias, que permite sua criação a partir de modelagem visual e realize raciocínio automático, deduzindo novos fatos e verificando fatos inconsistentes durante a criação das ontologias e apresentando estes resultados aos usuários. O objetivo foi demonstrado através de quatro experimentos (Vide Capítulo 4) realizados com nove participantes (Experimento 1, parte 1 e 2), seis participantes (Experimento 1, parte 3) e cinco participantes (Experimento 2).

Com a análise dos resultados, concluímos que o `Medina` possui capacidades de modelagem visual de ontologias com expressividade máxima $\mathcal{ALCHIQ}$, capacidade de realizar raciocínios automáticos e também capacidade de auxiliar os desenvolvedores no processo de criação de ontologias, reduzindo a dificuldade empregada no seu desenvolvimento. Além disso, o `Medina` proporciona aos usuários a possibilidade de aprender sobre ontologias e o significado das construções realizadas através da visualização das ontologias e da representação visual das inferências e inconsistências presentes na ontologia em desenvolvimento.

A seguir apresentamos um resumo das principais contribuições desta dissertação. Algumas limitações também são apresentadas e discutidas gerando também sugestões de possíveis trabalhos futuros.

6.1 CONTRIBUIÇÕES

Diante do contexto, a ferramenta desenvolvida proporcionará aos seus usuários, construir através de modelagem visual, ontologias expressivas e livres de fatos contraditórios, além de ser uma ferramenta interessante para o aprendizado de ontologias por parte de leigos interessados em seu desenvolvimento e uso. Assim será possível construir e obter como resultado axiomas com os principais construtores que a Lógica de Descrições (DL) admite, como [Horrocks, Peter e Patel-Schneider 2011]: conjunção ($\sqcap$), disjunção ($\sqcup$), negação ($\neg$), restrição existencial ($\exists r.C$), restrição universal ($\forall r.C$) e restrições de cardinalidade ($\leq \mid \geq \mid = n\ r.C$).

Identificamos como contribuições originais desta dissertação os seguintes avanços em relação ao estado da arte:

- Uma ferramenta de autoria inteligente, com arquitetura que permite, de fato, realizar raciocínio automático;

- Detecção e verificação de inconsistências em tempo de desenvolvimento nas bases de conhecimento construídas;
- Construção de bases de conhecimento (Ontologias) em OWL DL 2 com expressividade de alto nível a partir de modelagem visual;
- Auxiliar equipes e usuários interessados no desenvolvimento de ontologias a partir dos modelos visuais no processo de Engenharia de Conhecimento, reduzindo a dificuldade e o tempo empregados no desenvolvimento destas;
- Possibilidade de que usuários acelerem o desenvolvimento das bases de conhecimento modeladas como ontologias no qual são gerados automaticamente códigos OWL DL 2 a partir dos modelos visuais;
- Possibilidade de ser utilizada como uma ferramenta de ensino/aprendizagem de ontologias;

6.2 IDENTIFICAÇÃO DE TRABALHOS FUTUROS E LIMITAÇÕES

Vários pontos do trabalho podem ser melhorados. A seguir são apresentadas melhorias a limitações do presente trabalho:

- **Aumento da expressividade da modelagem visual:** Como vimos nos Trabalhos Relacionados (Vide Capítulo 5), o `Medina` é, entre as ferramentas verificadas, a ferramenta mais expressiva ($\mathcal{ALCHIQ}$), capaz de construir axiomas através de modelagem visual e apresentar raciocínio automático ao mesmo tempo. Mesmo com essa capacidade, é possível estender ainda mais a expressividade da ferramenta, permitindo o uso de *data properties*, enumeração, e características das propriedades (simetria, funcional e transitividade), buscando assim alcançar a expressividade máxima em relação a OWL DL, sempre levando em consideração a representação visual dos raciocínios processados.
- **Maneiras alternativas de lidar com os construtores universal e existencial:** Foi percebido na análise qualitativa da modelagem uma certa dificuldade dos usuários no uso desses construtores. Notações alternativas podem melhorar o tempo e a dificuldade ao modelar ontologias de maneira visual. A ligação direta entre a propriedade e a classe que fazem parte da restrição existencial ou universal, semelhante ao `Eddy` (LEMBO et al., 2016), pode ser uma possível abordagem para mitigar a dificuldade dos usuários no uso desses construtores.
- **Maneiras alternativas de representar classes complexas:** A combinação de técnicas de visualização permite que uma maior variedade de usuários possa utilizar o sistema proposto. Ocultar axiomas foi uma contribuição deste trabalho, porém

maneiras alternativas de representar podem ser acrescidas, como uma representação semelhante conjuntos, ou como proposta por Amaral (2008), mais relacionada ao significado e não a sintaxe OWL DL 2.

- **Melhorias na GUI do `Medina`:** Mesmo com uma boa avaliação por parte dos usuários em relação as suas funcionalidades, o `Medina` pode apresentar uma representação visual e uma tela mais elegante e compatível com as GUIs implementadas atualmente. Uma alternativa é a troca da tecnologia `JUNG`, implementada com a biblioteca de interface gráfica *Swing* da linguagem Java por tecnologias mais modernas de interface gráfica.

- **Implementação de uma ontologia para a validação da linguagem visual do `Medina`:** Na implementação do `Medina` foram desenvolvidos algoritmos voltados a validação e classificação de cada nó, atribuindo quais os arcos que podem ser conectados aos nós. Essa classificação de nós e a validação das relações entre esses nós pode ser implementada com ontologias, retirando a necessidade da implementação compilada por uma implementação declarada.

- **Integração com linguagem natural:** Atualmente o `Medina` suporta a criação de axiomas utilizando a *Manchester Syntax*. Outra possível funcionalidade é a integração do sistema com tecnologias de PLN (processamento de linguagem natural), permitindo que seus usuários criem axiomas a partir de linguagem natural novos axiomas e os ter representados na linguagem visual do `Medina`.

- **Integração com o OOPS!:** A qualidade da ontologia pode ser afetada pelas dificuldades envolvidas na sua modelagem, o que pode implicar o surgimento de anomalias em ontologias, levando à necessidade de validar ontologias, ou seja, avaliar sua qualidade e correção (POVEDA-VILLALÓN; SUÁREZ-FIGUEROA; GÓMEZ-PÉREZ, 2012). O OOPS![1] (*OntOlogy Pitfall Scanner!*) ajuda na detecção de algumas das armadilhas mais comuns que aparecem ao desenvolver ontologias. A integração dessa tecnologia com o `Medina` pode ser útil, representando de maneira visual, as anomalias presentes na ontologia obtidas pelo OOPS!, auxiliando no desenvolvimento de ontologias dos usuários.

- ***Plugin* do `Protégé`:** Sendo o `Protégé` o *software* mais utilizado por engenheiros de ontologias (MUSEN, 2015), uma contribuição futura é o desenvolvimento de um *plugin* para o `Protégé` que possibilite a construção de ontologias expressivas com raciocínio automático integrado ao `Protégé`, como uma aba adicional, por exemplo, representando assim uma diferente perspectiva de desenvolvimento no sistema.

[1] http://oops.linkeddata.es/

REFERÊNCIAS

AMARAL, F. N. do. Visualizing the semantics (not the syntax) of concept descriptions. In: **Proceedings of VI TIL**. Vila Velha, ES: [s.n.], 2008.

ANTONIOU, G.; GROTH, P.; HARMELEN, F. v. v.; HOEKSTRA, R. **A Semantic Web Primer**. Cambridge MA: The MIT Press, 2012.

BAADER, F. Description Logic Terminology. In: BAADER, F.; CALVANESE, D.; MCGUINNESS, D. L.; NARDI, D.; PATEL-SCHNEIDER, P. F. (Ed.). **The Description Logic Handbook: Theory, implementation and applications**. [S.l.]: Cambridge University Press, 2003.

BAADER, F.; CALVANESE, D.; MCGUINNESS, D.; NARDI, D.; PATEL-SCHNEIDER, P. **The Description Logic Handbook Theory, Implementation, And Applications.** [S.l.]: Cambridge University Press, 2003.

BAADER, F.; HORROCKS, I.; SATTLER, U. Description logics as ontology languages for the semantic web. In: **Mechanizing mathematical reasoning**. Berlin, Heidelberg: Springer, 2005. p. 228–248.

BAADER, F.; NUTT, W. Basic Description Logics. In: BAADER, F.; CALVANESE, D.; MCGUINNESS, D. L.; NARDI, D.; PATEL-SCHNEIDER, P. F. (Ed.). **The Description Logic Handbook: Theory, implementation and applications**. [S.l.]: Cambridge University Press, 2003.

BĀRZDIŅŠ, J.; BĀRZDIŅŠ, G.; ČERĀNS, K.; LIEPIŅŠ, R.; SPROĢIS, A.; OVČIŅŅIKOVA, J.; RIKAČOVS, S. OWLGrEd: a UML Style Graphical Editor for OWL. In: **ORES-2010 Ontology Repositories and Editors for the Semantic Web**. Hersonissos, Greece: [s.n.], 2010.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The Semantic Web. **Scientific America**, v. 284, n. 5, p. 34–43, 2001.

BRACHMAN, R.; LEVESQUE, H. **Knowledge Representation and Reasoning**. San Francisco, CA, USA: Morgan Kaufmann, 2004.

CAI, S.; MING, Z.; LI, S. BALC: A belief extension of description logic ALC. In: **9th International Conference for Young Computer Scientists**. Hunan, China: IEEE, 2008. p. 1711–1716.

CATENAZZI, N.; SOMMARUGA, L.; MAZZA, R. User-friendly ontology editing and visualization tools: the OWLeasyViz approach. In: **13th International Conference Information Visualisation**. Barcelona, Espanha: IEEE, 2009.

CONSOLE, M.; LEMBO, D.; SANTARELLI, V.; SAVO, D. F. Graphol: Ontology Representation through Diagrams. In: **Description Logics**. [S.l.: s.n.], 2014. p. 483–495.

DEITEL, P.; DEITEL, H. **Java how to program**. [S.l.]: Prentice Hall Press, 2011.

DENAUX, R. **Intuitive ontology authoring using controlled natural language**. Tese (Doutorado em Filosofia) — University of Leeds, 2013.

DENAUX, R.; THAKKER, D.; DIMITROVA, V.; COHN, A. G. Interactive semantic feedback for intuitive ontology authoring. In: **7th International Conference on Formal Ontology in Information Systems**. Graz, Austria: [s.n.], 2012. p. 160–173.

DIMITROVA, V.; DENAUX, R.; HART, G.; DOLBEAR, C.; HOLT, I.; COHN, A. G. Involving domain experts in authoring owl ontologies. In: **7th International Semantic Web Conference**. Karlsruhe, Alemanha: Springer, 2008. p. 1–16.

FILHO, D. M.; FREITAS, F.; OTTEN, J. RACCOON: A Connection Reasoner for the Description Logic $\mathcal{ALC}$. In: **21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning**. Maun, Botswana: EasyChair, 2017. v. 46, p. 200–211.

FREITAS, F.; OTTEN, J. A Connection Calculus for the Description Logic $\mathcal{ALC}$. In: **29th Canadian Conference on Artificial Intelligence**. Victoria, BC, Canadá: Springer, 2016. p. 243–256.

FREITAS, F.; STUCKENSCHMIDT, H.; NOY, N. F. Guest editor's introduction: Ontology issues and applications. **Journal of the Brazilian Computer Society**, v. 11, n. 2, p. 5–16, 2005.

FU, B.; NOY, N. F.; STOREY, M.-A. Indented tree or graph? A usability study of ontology visualization techniques in the context of class mapping evaluation. In: **12th International Semantic Web Conference**. Sydney, Austrália: [s.n.], 2013. p. 117–134.

GESTERKAMP, L.; REBSTADT, J.; MERTENS, R. Tackling Complex Ontologies with AVOnEd — Aspect-Oriented Visual Ontology Editor. In: **11th International Conference on Semantic Computing**. San Diego, Estados Unidos: [s.n.], 2017. p. 268–269.

GLIMM, B.; HORROCKS, I.; MOTIK, B.; STOILOS, G.; WANG, Z. HermiT: an OWL 2 reasoner. **Journal of Automated Reasoning**, Springer, v. 53, n. 3, p. 245–269, 2014.

GRAU, B. C.; HORROCKS, I.; MOTIK, B.; PARSIA, B.; PATEL-SCHNEIDER, P. F.; SATTLER, U. OWL2: The next step for OWL. **Journal of Web Semantics**, v. 6, n. 4, p. 309–322, 2008.

GRIMM, S.; HITZLER, P.; ABECKER, A. Knowledge representation and ontologies. **Semantic Web Services: Concepts, Technologies, and Applications**, Springer, p. 51–105, 2007.

GRUBER, T. R. Toward principles for the design of ontologies used for knowledge sharing? **International journal of human-computer studies**, Elsevier, v. 43, n. 5, p. 907–928, 1995.

HAARSLEV, V.; MÖLLER, R. Racer system description. In: **1st International Joint Conference on Automated Reasoning**. Siena, Itália: Springer, 2001. p. 701–705.

HALLAY, F.; HARTMANN, S.; KEWITZ, N.; MERTENS, R. An Aspect-Oriented Visual Ontology Editor with Edit-Time Consistency Checking. In: **11th International Conference on Semantic Computing**. San Diego, Estados Unidos: [s.n.], 2017. p. 297–304.

HORRIDGE, M.; BECHHOFER, S. The OWL API: A Java API for OWL Ontologies. **Semantic Web Journal**, IOS Press, v. 2, n. 1, p. 11–21, 2011.

KALYANPUR, A.; PARSIA, B.; HORRIDGE, M.; SIRIN, E. Finding all justifications of owl dl entailments. In: ABERER, K.; CHOI, K.-S.; NOY, N.; ALLEMANG, D.; LEE, K.-I.; NIXON, L.; GOLBECK, J.; MIKA, P.; MAYNARD, D.; MIZOGUCHI, R.; SCHREIBER, G.; CUDRÉ-MAUROUX, P. (Ed.). **The Semantic Web**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 267–280. ISBN 978-3-540-76298-0.

KATIFORI, A.; HALATSIS, C.; LEPOURAS, G.; VASSILAKIS, C.; GIANNOPOULOU, E. Ontology visualization methods—a survey. **ACM Computing Surveys (CSUR)**, ACM, v. 39, n. 4, p. 10, 2007.

KLEIN, M. Xml, rdf, and relatives. **IEEE Intelligent Systems**, IEEE, v. 16, n. 2, p. 26–28, 2001.

KRIVOV, S.; WILLIAMS, R.; VILLA, F. GrOWL: A tool for visualization and editing of OWL ontologies. **Web Semantics: Science, Services and Agents on the World Wide Web**, Elsevier, v. 5, n. 2, p. 54–57, 2007.

LEHMANN, J.; VOELKER, J. An introduction to ontology learning. **Perspectives on ontology learning. AKA/IOS Press, Heidelberg**, p. 9–16, 2014.

LEMBO, D.; PANTALEONE, D.; SANTARELLI, V.; SAVO, D. F. Eddy: A graphical editor for OWL 2 ontologies. In: **25th International Joint Conference on Artificial Intelligence**. New York, Estados Unidos: [s.n.], 2016. p. 4252–4253.

LIEBIG, T.; HALFMANN, M. Explaining subsumption in alehfr+ tboxes. In: **International Workshop on Description Logics**. Edinburgh, Escócia: [s.n.], 2005. p. 144–151.

LIEBIG, T.; NOPPENS, O. OntoTrack: A semantic approach for ontology authoring. **Web Semantics: Science, Services and Agents on the World Wide Web**, v. 3, n. 2-3, p. 116–131, 2005.

LIEPINŠ, R.; CERANS, K.; SPROGIS, A. Visualizing and Editing Ontology Fragments with OWLGrEd. In: **I-SEMANTICS (Posters & Demos)**. Graz, Austria: [s.n.], 2012. v. 932, p. 22–25.

LIEPINŠ, R.; GRASMANIS, M.; BOJARS, U. OWLGrEd ontology visualizer. In: **International Conference on Developers - Volume 1268**. Riva del Garda, Itália: CEUR-WS.org, 2014. p. 37–42.

MAEDCHE, A.; STAAB, S. Ontology Learning for the Semantic Web. **IEEE Intelligent Systems**, IEEE, v. 16, p. 72—79, 2001.

MCBRIDE, B. The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS. In: STAAB, S.; STUDER, R. (Ed.). **Handbook on Ontologies**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. p. 51–65. ISBN 978-3-540-24750-0.

MUSEN, M. A. The protégé project: a look back and a look forward. **AI matters**, ACM, v. 1, n. 4, p. 4–12, 2015.

NARDI, D.; BRACHMAN, R. J. An Introduction to Description Logics. In: BAADER, F.; CALVANESE, D.; MCGUINNESS, D. L.; NARDI, D.; PATEL-SCHNEIDER, P. F. (Ed.). **The Description Logic Handbook: Theory, implementation and applications**. [S.l.]: Cambridge University Press, 2003.

NEGRU, S.; LOHMANN, S. A Visual Notation for the Integrated Representation of OWL Ontologies. In: **9th International Conference on Web Information Systems and Technologies**. Aachen, Alemanha: SciTePress, 2013. p. 308–315.

O'MADADHAIN, J.; FISHER, D.; SMYTH, P.; WHITE, S.; BOEY, Y.-B. Analysis and visualization of network data using JUNG. **Journal of Statistical Software**, v. 10, n. 2, p. 1–35, 2005.

PARSIA, B.; SIRIN, E. Pellet: An OWL DL reasoner. In: **3rd International Semantic Web Conference**. Hiroshima, Japão: [s.n.], 2004.

POVEDA-VILLALÓN, M.; SUÁREZ-FIGUEROA, M. C.; GÓMEZ-PÉREZ, A. Validating ontologies with OOPS! In: **The 18th International Conference on Knowledge Engineering and Knowledge Management**. Galway, Irlanda: Springer, 2012. p. 267–281.

REBSTADT, J.; BRINKSCHULTE, L.; ENDERS, A.; MERTENS, R. A Visual Language for OWL Lite Editing. In: **SEMANTiCS (Posters, Demos, SuCCESS)**. Leipzig, Alemanha: CEUR-WS.org, 2016.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009. ISBN 0136042597, 9780136042594.

SANTANA, F.; SCHOBER, D.; MEDEIROS, Z.; FREITAS, F.; SCHULZ, S. Ontology patterns for tabular representations of biomedical knowledge on neglected tropical diseases. **Bioinformatics**, Oxford University Press, v. 27, n. 13, p. i349–i356, 2011.

SARKER, M. K.; KRISNADHI, A. A.; HITZLER, P. OWLAx: A protégé plugin to support ontology axiomatization through diagramming. In: **15th International Semantic Web Conference**. Kobe, Japão: [s.n.], 2016.

SHADBOLT, N.; BERNERS-LEE, T.; HALL, W. The semantic web revisited. **IEEE intelligent systems**, IEEE, v. 21, n. 3, p. 96–101, 2006.

TSARKOV, D.; HORROCKS, I. FaCT++ description logic reasoner: System description. In: **3rd International Joint Conference**. Seattle, Estados Unidos: Springer, 2006. p. 292–297.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in software engineering**. [S.l.]: Springer Science & Business Media, 2012.

APÊNDICE A – QUESTIONÁRIOS

Neste apêndice estão inseridos todos os questionários utilizados nos experimentos (Vide Capítulo 4).

A.1 QUESTIONÁRIO 1

1. De acordo com os modelos visuais apresentados com uso do Medina, qual foi o nível de percepção do possível raciocínio automático de Subsunção e Inconsistência ocorrido:

- ○ Fácil perceber onde ocorreria subsunção e inconsistência
- ○ Razoável perceber onde ocorreria subsunção e inconsistência
- ○ Difícil perceber onde ocorreria subsunção e inconsistência
- ○ Não conseguiu perceber onde ocorreria subsunção e inconsistência

Motivo da resposta:

2. De acordo com os modelos (Fragmentos de Bases de Conhecimento) apresentados na notação em DL, qual foi o nível de percepção do possível Raciocínio Automático de subsunção e inconsistência ocorrido?

- ○ Fácil perceber onde ocorreria subsunção e inconsistência
- ○ Razoável perceber onde ocorreria subsunção e inconsistência
- ○ Difícil perceber onde ocorreria subsunção e inconsistência
- ○ Não conseguiu perceber onde ocorreria subsunção e inconsistência

Motivo da resposta:

3. De acordo com os modelos (Fragmentos de Bases de Conhecimento) apresentados, apenas em OWL DL, qual foi o nível de percepção do possível raciocínio automático de Subsunção e Inconsistência ocorrido?

- ◦ Fácil perceber onde ocorreria subsunção e inconsistência
- ◦ Razoável perceber onde ocorreria subsunção e inconsistência
- ◦ Difícil perceber onde ocorreria subsunção e inconsistência
- ◦ Não conseguiu perceber onde ocorreria subsunção e inconsistência

Motivo da resposta:

4. De acordo com os modelos (Fragmentos de Bases de Conhecimento) apresentados com uso do Protégé, qual foi o nível de percepção do possível raciocínio automático de Subsunção e Inconsistência ocorrido:

- ◦ Fácil perceber onde ocorreria subsunção e inconsistência
- ◦ Razoável perceber onde ocorreria subsunção e inconsistência
- ◦ Difícil perceber onde ocorreria subsunção e inconsistência
- ◦ Não conseguiu perceber onde ocorreria subsunção e inconsistência

Motivo da resposta:

5. De acordo com as formas de verificar o raciocínio automático ocorrido nos fragmentos [das Bases de Conhecimento] construídos com Modelagem Visual a partir do Medina, OWL DL a partir da notação em DL, Código OWL DL e com uso do Protégé, qual foi a melhor forma?

   OBS.: Ranquear as formas em 1º, 2º, 3ª, 4ª e 5º lugar.

___ Com a notação DL
___ Com o Código OWL DL
___ Com a modelagem visual do `Medina`
___ Com a construção no `Protégé`
___ Nenhuma das formas

Motivo da resposta:

6. De acordo com as formas de aprender sobre o raciocínio automático ocorrido nos fragmentos [das Bases de Conhecimento] construídos com Modelagem Visual a partir do Medina, OWL DL a partir da notação em DL, Código OWL DL e com uso do Protégé, qual foi a melhor forma?

   OBS.: Ranquear as formas em 1º, 2º, 3ª, 4ª e 5º lugar.

___ Com a notação DL
___ Com o Código OWL DL
___ Com a modelagem visual do `Medina`
___ Com a construção no `Protégé`
___ Nenhuma das formas

Motivo da resposta:

A.2 QUESTIONÁRIO 2

Qual a ferramenta utilizada para a modelagem da ontologia?

1. Qual a sua avaliação em relação a encontrar as funcionalidades necessárias para modelar a ontologia:

   Muito Fácil ○ ○ ○ ○ ○ Muito Difícil

2. Qual a sua avaliação em relação a modelagem da ontologia:

Muito Simples e de Fácil Compreensão ○ ○ ○ ○ ○ Muito complicada e de Difícil Compreensão

3. Qual a sua avaliação em relação as inferências sobre a ontologia modelada através da ferramenta:

   Muito Útil ○ ○ ○ ○ ○ Irrelevante

4. Qual a sua avaliação em relação a visualização das insatisfatibilidades de classes sobre a ontologia modelada através da ferramenta:

   Muito Útil ○ ○ ○ ○ ○ Irrelevante

5. Sugestões/Comentários:

A.3 QUESTIONÁRIO 3

1. A modelagem visual apresentada pelo Medina é adequada as minhas necessidades.

- ◦ Concordo plenamente
- ◦ Concordo parcialmente
- ◦ Não concordo nem discordo
- ◦ Discordo parcialmente
- ◦ Discordo Totalmente

2. A visualização de deduções apresentada pelo Medina é adequada para as minhas necessidades.

- ◦ Concordo plenamente
- ◦ Concordo parcialmente
- ◦ Não concordo nem discordo
- ◦ Discordo parcialmente
- ◦ Discordo Totalmente

3. A funcionalidade de ocultar axiomas apresentada pelo Medina é adequada as minhas necessidades

- ◦ Concordo plenamente
- ◦ Concordo parcialmente
- ◦ Não concordo nem discordo
- ◦ Discordo parcialmente
- ◦ Discordo Totalmente

4. Se possível, apresente pontos positivos da ferramenta.

5. Se possível, apresente pontos negativos da ferramenta.