Universidade do Minho
Escola de Engenharia
Departamento de Informática

André Filipe Amorim Lara

Visualization of Ontology Evolution using OntoDiffGraph

February 2018

Universidade do Minho
Escola de Engenharia
Departamento de Informática

André Filipe Amorim Lara

Visualization of Ontology Evolution using OntoDiffGraph

Master dissertation
Master Degree in Computer Science

Dissertation supervised by
Pedro Rangel Henriques
Alda Lopes Gançarski

February 2018

ABSTRACT

Ontologies evolve with the passing of time due to improvements, corrections or changes in requirements that need to be made. It is hard to keep track of these changes made in an ontology without a tool built specifically for that purpose.

The goal of this master's work is the creation of a visualization technique with the objective of allowing the viewer to easily identify changes made in an ontology by comparing two versions of it.

The proposed approach adapts the already existing VOWL (Visual Notation for OWL Ontologies) specification so that it can also display the differences between two versions of an ontology through the use of a graph. This was implemented in an application, called OntoDiffGraph, however this feature is not all that this application has implemented. In fact, to be able to detect the changes that were made, an algorithm was developed that allows the application to find the axioms that were added or removed in an ontology and display them in a graph.

OntoDiffGraph also contains several other features that help with the identification of the changes that were made, such as displaying all axioms of the ontology in a list or filtering nodes and edges of the graph that are not relevant to the visualization of changes made in the ontology.

OntoDiffGraph was used in an experiment to obtain user feedback and discover how it performed when compared to a similar application (OWLDiff). The participants gave a lot of important constructive feedback and they also displayed a preference of OntoDiffGraph over the other alternative.

RESUMO

As ontologias evoluem com o passar do tempo devido a melhoramentos, correções ou mudanças nos requisitos que necessitam de ser feitas. É difícil observar estas mudanças feitas numa ontologia sem utilizar ferramentas especializadas para este fim.
O objetivo deste trabalho de mestrado é a criação de uma técnica de visualização com a intenção de oferecer ao observador a capacidade de fácil identificação das mudanças feitas numa ontologia ao comparar duas versões distintas.
A abordagem tomada adapta a especificação VOWL (Visual Notation for OWL Ontologies) de forma a que também seja possível visualizar as diferenças entre duas versões de uma ontologia através do uso de um grafo. Esta abordagem foi implementada numa aplicação, com o nome de OntoDiffGraph, no entanto esta funcionalidade não é tudo o que foi implementado. Para que seja possível detetar as mudanças que foram feitas, um algoritmo foi desenvolvido que permite que a aplicação, além de identificar os conceitos/relações e propriedades que sofreram mudanças, encontre os axiomas que foram adicionados ou removidos da ontologia e os apresente num grafo.
OntoDiffGraph também tem diversas outras funcionalidades que ajudam na identificação de mudanças, tal como a apresentação de todos os axiomas da ontologia numa lista ou a filtração de nodos e arestas que não são relevantes para a visualização das mudanças feitas na ontologia.
OntoDiffGraph foi também utilizado num experimento com o objetivo de obter críticas e comparar o seu desempenho com uma aplicação similar (OWLDiff). Os participantes no experimento ofereceram bastantes críticas construtivas e demonstraram uma preferência por OntoDiffGraph em vez de OWLDiff.

CONTENTS

1 INTRODUCTION 1
- 1.1 Objectives 2
- 1.2 Research Hypothesis 2
- 1.3 Document Organization 2

2 STATE OF THE ART 4
- 2.1 Background 4
- 2.2 Related work 6
  - 2.2.1 Ontology Visualization 6
  - 2.2.2 Change detection 10
  - 2.2.3 Change visualization 11

3 ONTODIFFGRAPH: THE PROPOSAL 12
- 3.1 Proposed notation 12
  - 3.1.1 Classes 13
  - 3.1.2 Object Properties 13
  - 3.1.3 Data Properties 13
  - 3.1.4 Annotation Properties 13
  - 3.1.5 Datatypes 14
  - 3.1.6 Imported Classes and Properties 14
  - 3.1.7 Anonymous Elements 14
  - 3.1.8 Element Characteristics 15
  - 3.1.9 Other 15
  - 3.1.10 Difference Notation 17
- 3.2 OntoDiffGraph: Architecture 19
- 3.3 Technology Choice 20
  - 3.3.1 OWL API 20
  - 3.3.2 Java 20
  - 3.3.3 JavaFX 20

4 ONTODIFFGRAPH: DEVELOPMENT 21
- 4.1 Graph interactivity 21
- 4.2 Graph layout 21
- 4.3 Ontology elements list 23
- 4.4 Element data 23
- 4.5 Axiom difference 24
- 4.6 Graph customization 24

4.7 Change detection 25
4.8 Visualization 26
5 CASE STUDY 28
6 EXPERIMENT 34
6.1 Experiment setup 34
6.2 Results 35
6.2.1 FOAF Ontology 35
6.2.2 MOD Ontology 36
6.2.3 Feedback 36
6.3 Discussion 39
7 CONCLUSION 41
7.1 Contributions 41
7.2 Future Work 42

A DOCUMENTATION TO SUPPORT THE EXPERIMENT 46
A.1 OntoDiffGraph and OWLDiff guide 46
A.2 Examples and questions 54

LIST OF FIGURES

Figure 1 Undirected graph 4
Figure 2 Directed weighted graph 4
Figure 3 Document class from the FOAF Ontology 5
Figure 4 Visualization of the ontology using WebVOWL 7
Figure 5 Visualization of an ontology with the VOWL notation 8
Figure 6 Graph notation for classes 13
Figure 7 Graph notation for object properties 13
Figure 8 Graph notation for data properties 13
Figure 9 Graph notation for annotation properties 13
Figure 10 Graph notation for datatypes 14
Figure 11 Graph notation for imported classes 14
Figure 12 Graph notation for imported properties 14
Figure 13 Graph notation for anonymous classes 14
Figure 14 Graph notation for anonymous object properties 15
Figure 15 Graph notation for a functional object property 15
Figure 16 Graph notation for the domain of an object property 15
Figure 17 Graph notation for the range of a data property 15
Figure 18 Graph notation to display a subclass connection 16
Figure 19 Graph notation to display a sub object property connection 16
Figure 20 Graph notation to display a equivalent class connection 16
Figure 21 Graph notation to display a equivalent object property connection 16
Figure 22 Graph notation to display an inverse object property connection 16
Figure 23 Graph notation to display a disjoin classes connection 17
Figure 24 Graph notation to display a disjoint union connection 17
Figure 25 Graph notation to display creation and removal of a class 17
Figure 26 Graph notation to display creation and removal of an object property 17
Figure 27 Graph notation to display the removal of the *disjointUnion* axiom 18
Figure 28 Graph notation to display changes in domains 18
Figure 29 Architecture of the OntoDiffGraph system 19
Figure 30 Tabs with all ontology elements 23
Figure 31 Tab showing information about a node 24
Figure 32 Tab showing the changes in axioms 24

Figure 33	Layout options	25
Figure 34	FOAF ontology displayed in OntoDiffGraph	27
Figure 35	Creating the ontology difference	28
Figure 36	Select older ontology version	29
Figure 37	Select most recent ontology version	30
Figure 38	Tick the "Show Minimal View" box	30
Figure 39	Elements of the ontology	31
Figure 40	Graph displaying some changes in the MOD ontology	31
Figure 41	List of axioms of the ontology	32
Figure 42	Participant responses for the statement	37
Figure 43	Participant responses for the statement	38
Figure 44	Participant responses for the statement	38
Figure 45	Compare Ontologies feature	48
Figure 46	Check for existing plugins	49
Figure 47	OWL Difference Plugin	50
Figure 48	Displaying differences in FOAF ontology	51
Figure 49	OntoDiffGraph User Interface	52

LIST OF ABBREVIATIONS

OntoDiffGraph	The application that was developed for this thesis.
FOAF	Friend Of A Friend, an ontology on the domain of social networks of human collaboration, friendship and association.
Java	An object oriented programming language. Java code is compiled to bytecode that is then run in the Java Virtual Machine, this allows easy portability of the code to several different systems.
JavaFX	A native Java library used to create desktop applications. It was created with the intent of replacing the already existing Swing library.
MOD	Metadata for Ontology Description and Publication, an ontology on the domain of ontology metadata.
Ontology	A formal method to define the structure of knowledge about a domain.
OWL	Web Ontology Language, a family of languages that are used to represent ontologies.
OWL API	A Java library used to manipulate OWL Ontologies.
OWLDiff	A Protégé plugin used to find differences in versions of an ontology.
Protégé	An application used to manipulate ontologies.
RDF	Resource Description Framework, a framework that describes resources in the Internet using XML.
RDFS	Resource Description Framework Schema, an extension to the already existing RDF vocabulary.

VOWL	Visual Notation for OWL Ontologies, a notation that details how to represent an ontology using a graph.

1 INTRODUCTION

Through the use of ontologies it is possible to store entities and their relations with each other. However with the increase in complexity of an ontology, the risk of the user becoming unable to keep up with the changes that are made might make it necessary to rethink the method used to visualize the ontology. One of the most intuitive forms of displaying an ontology is through the use of a graph, however there are various different forms to display the same information in a graph. Graphs can be presented through the use of force-directed, orthogonal, radial, trees, and many other types of layouts as it can be seen in Di Battista et al. (1994). In order to map ontology elements to graphical entities, different notations exist, one example of this is the Visual Notation for OWL Ontologies (Lohmann et al., 2014c), usually abbreviated to VOWL.

The Friend of a Friend Ontology (FOA), usually abbreviated to FOAF, is an ontology that describes people and the connections they have between each other. Since the year this ontology was created (2000) until the release of the most recent version (2014), this ontology went through several different versions. Tools to analyze this evolution in the ontology already exist (Kremen et al., 2011; Noy et al., 2002; Hartung et al., 2012). However the visualization aspect of these tools can still be greatly improved. The existence of these tools confirms that there is a problem that needs to be solved. The creation of a visualization technique that allows users to easily identify changes would help users analyze how ontologies are evolving and quickly determine what has been changed between different versions of an ontology.

With the use of a graph to display an ontology it is possible to display the results of a structural *diff* inside the graph. If this is successfully executed it is possible to create a way to easily display the evolution of an ontology. The existence of an application that implements these features will allow users to analyse the structural changes made in different versions of an ontology and visualize the impact that these changes made in the graph displaying all concepts and relationships of the ontology.

1.1 OBJECTIVES

This thesis has the following objectives:

- Proposal of a visualization technique that allows the users to easily identify changes in an ontology. Our contribution focuses on the possibility for the users to visualize the changes made on an ontology without having to see the entire ontology, which is difficult to manage.

- Implementation of this visualization technique in an application using Java.

1.2 RESEARCH HYPOTHESIS

This thesis research hypothesis is that through the use of the proposed technique, i. e. a visual *diff* using a graph to display the ontology and the changes made on it, it will be easier and faster to identify and locate the differences in versions of an ontology and help the user comprehend their meaning and impact.

1.3 DOCUMENT ORGANIZATION

This dissertation is divided in the following chapters:

- Introduction (Page 1): In Chapter 1 the reader is introduced to the context of this dissertation and what it aims to achieve.

- State of the art (Page 4): In Chapter 2 the state of the art in the different areas involved in this work is presented, including works on ontology visualization, change detection in ontologies, and the visualization of these changes.

- OntoDiffGraph: The Proposal (Page 12): In Chapter 3 it is discussed the proposed technique and the architecture of the application where it is going to be implemented, OntoDiffGraph.

- OntoDiffGraph: Development (Page 21): Chapter 4 contains information related to features and decisions that were taken during the development of the application.

- Case Study (Page 28): Chapter 5 contains a case study where we use the MOD ontology to show how a user would use OntoDiffGraph to find the differences between versions of this ontology.

- Experiment (Page 34): Chapter 6 shows the experiment that was conducted in order to test how OntoDiffGraph would perform when compared to another application

(OWLDiff) made with the same purpose, the detection of changes in different versions of an ontology.

- Conclusion (Page 41): In Chapter 7, it is analyzed what has been described in this dissertation and the future work that still can be done.

2

STATE OF THE ART

2.1 BACKGROUND

A graph is a structure composed of vertices and edges. Vertices usually represent an entity and the edges represent connections between these entities.
The graph can be very simple and only contain nodes and the edges as seen in Figure 1.

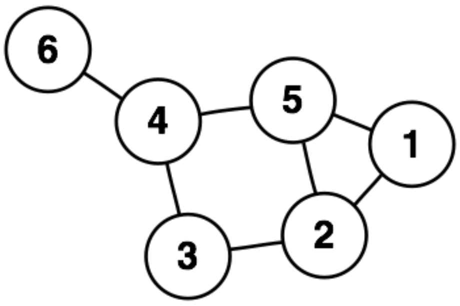


Figure 1.: Undirected graph

However a graph can contain a lot more information. In Figure 2 the graph contains edges that also display their direction and have a label showing the weight of that connection.

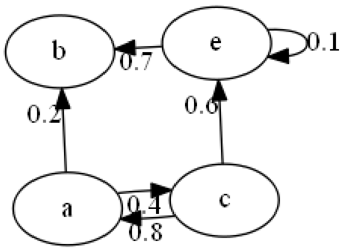


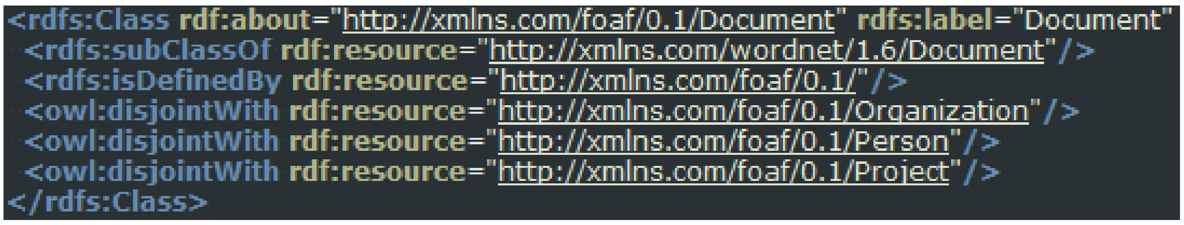
Figure 2.: Directed weighted graph

Choosing the best type of graph depends on the information we want to display. We can use undirected graphs or directed graphs depending on the relationships between the nodes. The edges can also be labeled if there is a need to distinguish relationships between

different edges, however in some graphs this is not important, as is the case in a weighted graph. In this type of graph what needs to be attached to the edge is a cost or the weight of that connection. This type of graph is usually used to find the optimal path between certain vertices in an attempt to reduce costs of inefficient travel.
An important issue concerning graph visualization is how vertices and edges should be placed in order to make the graph easy to understand. This is a very complex topic with various solutions and algorithms, some only suited for specific types of graphs. Some popular layout algorithms are force-directed (moving the vertices according to a system of forces), orthogonal (edges of the graph only bend in multiples of 90°) or a tree layout (best suited for acyclic graphs).

Another topic related to this dissertation is ontologies. An ontology is a formal description of concepts in a domain.

There are currently a lot of ontologies available in various domains. To describe people and their relationships on the web we have the FOAF Ontology, to describe products sold online there is the Good Relations Ontology and there is The Music Ontology that contains information related to the music industry, in the biology domain there is an ontology that intends to unify the representation of genes named the Gene Ontology.



```
<rdfs:Class rdf:about="http://xmlns.com/foaf/0.1/Document" rdfs:label="Document"
 <rdfs:subClassOf rdf:resource="http://xmlns.com/wordnet/1.6/Document"/>
 <rdfs:isDefinedBy rdf:resource="http://xmlns.com/foaf/0.1/"/>
 <owl:disjointWith rdf:resource="http://xmlns.com/foaf/0.1/Organization"/>
 <owl:disjointWith rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
 <owl:disjointWith rdf:resource="http://xmlns.com/foaf/0.1/Project"/>
</rdfs:Class>
```

Figure 3.: Document class from the FOAF Ontology

These structures are usually composed of the following elements:

- Classes, abstract objects that are used to represent concepts;
- Attributes, information that is associated with classes;
- Relations, used to connect the different classes and attributes together;
- Logical elements, these are used to apply restrictions and assertions to the elements of ontologies;
- Individuals, instances of classes that create the knowledge base.

By observing the elements used to create an ontology we can recognize that its structure can be represented through a graph. By representing classes and attributes as vertices and relations as edges we can display the structure of an ontology in the form of a directed graph. Since ontologies are stored as text, trying to read them without a specialized tool can be a daunting task. To help solving this problem, there is a high number of tools to ease the visualization and edition of an ontology, being the transformation into a graph one of the best ways to display the ontology to a user, as it allows him to easily see the vertices and the connections between them incredibly fast.

2.2 RELATED WORK

2.2.1 *Ontology Visualization*

There are languages that can be used to serialize the content and structure of an ontology. One of these languages is the Web Ontology Language (OWL (Dean and Schreiber, 2004)), built on the already existing Resource Description Framework (RDF (Beckett, 2004)) and RDF Schema (RDFS (Guha and Brickley, 2014)) specifications. OWL gives its users more expressive power than the RDF and RDFS specifications, allowing them to add more complexity to the information contained in an ontology by using chained properties, restrictions, defining the cardinality of relations and many others.

The following ontology is written with OWL using the RDF/XML Syntax:

```xml
<?xml version="1.0"?>
<rdf:RDF xmlns="http://www.semanticweb.org/myOntology#"
     xml:base="http://www.semanticweb.org/myOntology"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     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.semanticweb.org/myOntology"/>

    <owl:ObjectProperty rdf:about="http://www.semanticweb.org/myOntology#isFriendOf">
        <rdfs:domain rdf:resource="http://www.semanticweb.org/myOntology#Person"/>
        <rdfs:range rdf:resource="http://www.semanticweb.org/myOntology#Person"/>
    </owl:ObjectProperty>

    <owl:ObjectProperty rdf:about="http://www.semanticweb.org/myOntology#owns">
        <rdfs:domain rdf:resource="http://www.semanticweb.org/myOntology#Person"/>
        <rdfs:range rdf:resource="http://www.semanticweb.org/myOntology#Pet"/>
    </owl:ObjectProperty>

    <owl:Class rdf:about="http://www.semanticweb.org/myOntology#Person"/>

    <owl:Class rdf:about="http://www.semanticweb.org/myOntology#Pet"/>
</rdf:RDF>
```

Listing 2.1: Ontology written using RDF/XML Syntax

This ontology contains two owl:Classes (*Person* and *Pet*) and two owl:ObjectProperty (*isFriendOf* and *owns*). Classes represent the concepts and object properties represent the relations between concepts. These relations can be seen in the domain and range of the object properties. Inside the *owns* object property we can observe that *Person* is the domain and *Pet* is the range, this relation is read as Person owns Pet. In the other object property we can find the relation Person isFriendOf Person. The relations display to the user that persons can own pets and that persons can be friends with each other.

By observing the elements used to create an ontology we can recognize that its structure can be represented with a graph. By representing classes and attributes as vertices and relations as edges we can display the structure of an ontology in the form of a directed graph. Since serialized ontologies are hard for a user to read, there are a high number of tools to ease the visualization and edition of an ontology. The transformation into a graph that can be visualized is one of the best ways to display the ontology to a user, as it allows him to easily see the various ontology elements and their connections incredibly fast. This can be observed in Figure 4, where the graph contains the OWL ontology that was displayed previously.

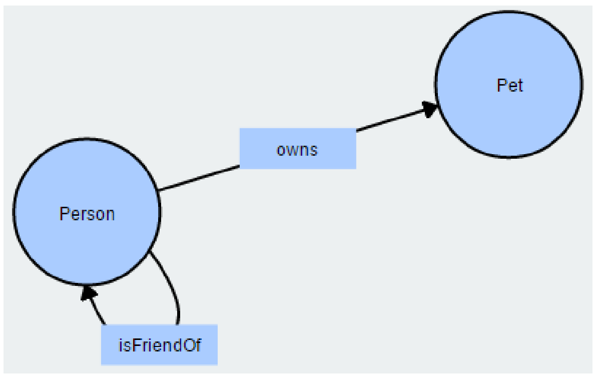


Figure 4.: Visualization of the ontology using WebVOWL

Visual Notation for OWL Ontologies (VOWL) (Lohmann et al., 2014c) is a visual language for ontologies with the aim to help users understand the structure of an ontology intuitively. The VOWL notation provides a graphical version of the various existing ontology elements and has been improved since its initial release.
In Figure 5 it is possible to observe the representation of the various elements of an ontology. The blue circles represent classes and the blue and green boxes represent the object and data properties of the ontology. It is possible to see that the object property *OPropC* is a functional property, *OPropA* and *OPropAA* are inverses of each other and that *OPropB* is connected to *ClassC* (Domain) and *Thing* (Range). The green boxes represent data properties and the yellow boxes contain the datatype of the data property. It is also possible to notice that *ClassAA* has a different visual representation than the other classes, this is because *ClassAA*

and *ClassA* are equivalent classes. Because of this they are represented in the graph by the same circle. This notation can be read in more detail in Lohmann et al. (2014c).

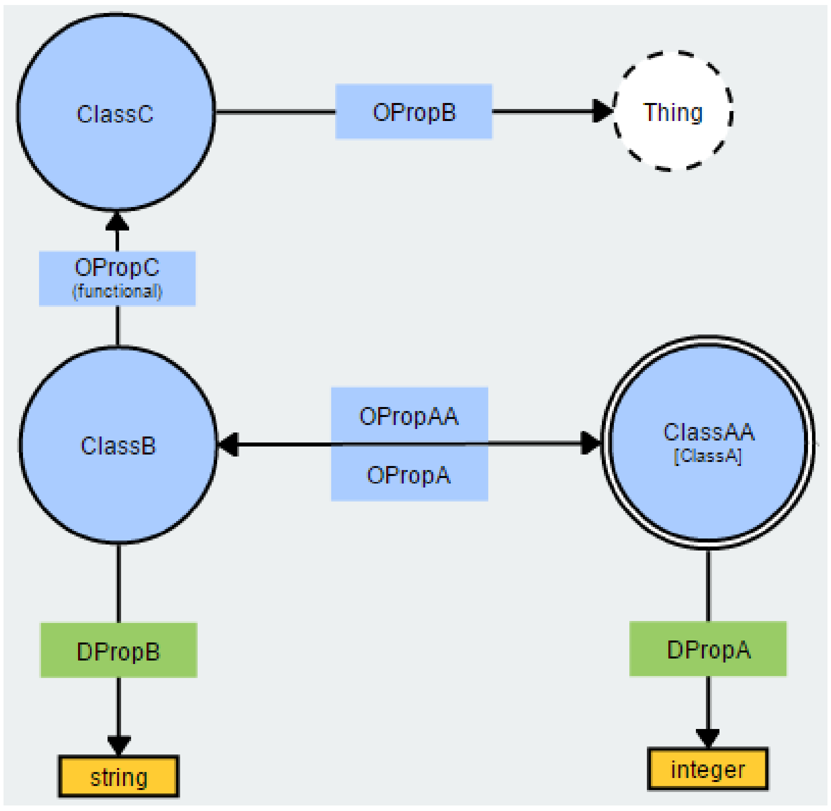


Figure 5.: Visualization of an ontology with the VOWL notation

There are several tools that are already using this notation:

- WebVOWL, a web application that implements the VOWL notation using a force-directed graph layout (Lohmann et al., 2014a).
- ProtégéVOWL, a Protégé plug-in that allows its users to visualize the ontology using a force-directed graph layout (Lohmann et al., 2014b).
- QueryVOWL, a visual query language that uses the VOWL notation for visualization (Haag et al., 2015).
- OntoBench, used to generate benchmark ontologies, allows the visualization of an ontology through the use of WebVOWL (Link et al., 2016).

OntoViBe (Haag et al., 2014) is an ontology for visualization benchmarking. This ontology was created with the aim of assisting the testing of ontology visualizations by allowing the visualization of most of the elements defined by the OWL 2 specification with the objective of checking if these various elements are correctly displayed. This ontology is used to test the visualization of classes, properties and datatypes of an ontology, but it should not be used to test scalability of the visualization or the representation of individuals in the ontology.

There is also a very wide selection of ontology visualization tools using different techniques to display the structure of the ontology:

- TGViz (Alani, 2003) is a Protégé plug-in that displays ontologies in a graph and allows the user to search and customize the notation of the nodes.
- NavigOWL (Hussain et al., 2014) is a Protégé plug-in that displays ontologies in a graph with various types of layout drawing techniques.
- OWLViz, is a Protégé plug-in that generates graphs through the use of GraphViz, however the generated graphs are static.
- KC-Viz (Motta et al., 2011) was a NeOn Toolkit plugin (currently it is integrated in the core). It uses an algorithm to identify the key concepts of an ontology by scoring them in several criteria that are then added in a weighted sum to get the final score. When it is displayed with a graph, the nodes of the classes chosen to be most important are displayed in a larger size than the others.
- OntoTrix (Bach et al., 2011) displays the implementation of a technique to visualize populated ontologies through the use of adjacency matrices and graphs. This technique was inspired by NodeTrix (Henry et al., 2007).
- Jambalaya (Storey et al., 2001) is a Protégé plugin that implements a visualization technique named SHriMP where the classes are nested inside other classes according to their class hierarchy. However this plugin can also display the class hierarchy through the use of several other layouts (radial, spring, grid and tree layouts).
- Knoocks (Kriglstein and Wallner, 2010) is an application that contains blocks (classes directly connected to owl:Thing) and the subclasses of these classes are contained inside the corresponding block. Inside the block, the subclasses are placed to the right of their predecessor in order to easily understand the hierarchy of the ontology.
- CropCircles (Parsia et al., 2005) is part of the SWOOP Ontology Editor and it provides a very minimalistic technique to view the hierarchy of an ontology. The classes of the ontology are displayed as circles where the subclasses are contained inside their parent. The name of the class is only displayed inside a tooltip, making the representation of the structure simple and compact.
- OWLGrEd (Barzdins et al., 2010) can be used as a desktop application or online. This application uses an extension of the UML notation to represent the ontologies and

allows the users to edit the ontology contents and filter the content they do not want to see.

- OntoSphere 3D (Bosca et al., 2005) is a Protégé plugin that uses a 3-dimensional space to display ontologies. To visualize the ontology the user uses several different views with varying level of detail, however this visualization technique has problems displaying ontologies with large sizes.

2.2.2 *Change detection*

COnto-Diff (Complex Ontology Diff) is an algorithm developed to detect the changes between two ontologies (Hartung et al., 2013). The algorithm detects changes based on insertion, update and delete operations and then transforms these basic operations into more complex ones such as merge, split or map. This algorithm has been implemented in CODEX (Complex Ontology Diff Explorer), an application that allows its users to identify what was modified between two ontologies, through the use of the COnto-Diff algorithm (Hartung et al., 2012).

PromptDiff is an algorithm that detects changes between two ontologies (Noy et al., 2002). The Protégé plugin called PromptDiff verifies the correctness of this algorithm. This algorithm uses matchers to find what has changed in the ontology, the results from a matcher are used as input for the other matchers, all the matchers continue to process the ontologies until they can't detect more changes.
This algorithm is used by a group of several Protégé plugins, called Prompt suite, that contains the following elements:

- iPROMPT, an ontology merging tool.
- AnchorPROMPT, a tool used to find similarities between ontologies.
- PROMPTFactor, used to extract parts of an ontology.
- PROMPTDiff, used to find and display the differences between ontologies.
- PROMPT-Viz, used to visualize the results obtained by PROMPTDiff.

CEX is an algorithm that can find the logical difference between two $\mathcal{EL}$ (Existential Language) terminologies (Konev et al., 2008). This algorithm has been implemented in OWLDiff (Kremen et al., 2011), a practical tool for comparison and merge of two ontologies. This application uses two different algorithms to detect changes, a simple one to detect add/update/delete operations and the CEX algorithm to detect complex effects caused by these differences.

2.2.3 *Change visualization*

PROMPT-Viz (Perrin, 2004) is a Protégé plugin that extends the PROMPTDiff plugin so that the differences between ontologies can be displayed intuitively. This plugin uses a zoomable treemap layout to optimize the utilization of the screen, facilitating the visualization of large ontologies. The classes in the treemap are represented by nodes and the changes in the ontology are displayed through the use of different colours. The colours used to display the difference between ontologies are white, gray, red, green, yellow and blue. This plugin also contains a filter to hide unimportant information in order to easily visualize the most relevant information.
With PROMPT-Viz, arcs are used to display changes in the location of classes in the treemap and its connections to the other nodes. The selection of a node reveals its arcs to other nodes, these arcs are coloured depending on how the destination class has been changed. This feature cluttered the visualization if it was active for all the nodes all the time, therefore this information is only displayed on a selection event. However the users that evaluated this plugin still had difficulty understanding the information the arcs were trying to convey.

AberOWL (Rodrıguez-Garcıa et al.) is an ontology repository and framework for ontology-based data access through the use of a web interface. In AberOWL it is possible to visualize ontologies as directed graphs where nodes represent classes and edges represent axioms. It is also possible to visualize the difference between several versions of an ontology, to achieve this, AberOWL uses a different colour for each version of the ontology to differentiate them. However the visualization of the evolution of an ontology is not the main feature of this system. It is not possible to search for the changes between two different versions, the user must expand the tree manually to visualize the entire ontology and be able to detect the evolution, this is not a practical solution because if the user needs to visualize a large ontology with a large number of nodes, it will take a very long time to get the desired results.

3 ONTODIFFGRAPH: THE PROPOSAL

The main challenge in the creation of the visualization technique is the risk of overloading the user with too much information. An ontology can contain a lot of different elements that need to be presented with different types of nodes and edges in a graph. If the specification chosen to transform the ontology into a graph is not appropriate, the user might be unable to easily understand what is being displayed. The technique this thesis aims to develop needs to add a new layer of graphical information (difference between the ontologies) to the already existing graph. Therefore the decisions made in the development process need to consider the existence of this risk.

We propose a system, called OntoDiffGraph, exploring a new visualization technique to easily identify the differences between two versions of an ontology. This technique uses a visual notation that is based on the already existing VOWL specification (Lohmann et al., 2014c), used to map ontology elements to visual graph elements in order to easily display an ontology.

3.1 PROPOSED NOTATION

The proposed notation is an extension of the VOWL notation. However, some of the elements had to be simplified or adapted in order to correctly be able to display the difference between versions of an ontology. An example of this are the object properties, that are labels of edges in the VOWL notation, however in this proposed notation they had to be changed to nodes of the graph.

The decision to use VOWL was due to it being a well documented and simple notation that was already being used by several applications that needed to display ontologies using a graph.

3.1.1 *Classes*

Classes are represented as blue circles with a black border and their class name inside as seen in Figure 6.

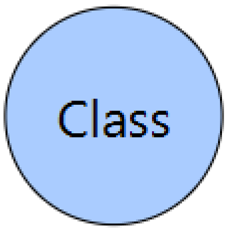


Figure 6.: Graph notation for classes

3.1.2 *Object Properties*

Object Properties are represented as blue rectangles with their property name inside as seen in Figure 7.

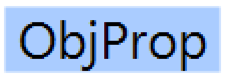


Figure 7.: Graph notation for object properties

3.1.3 *Data Properties*

Data Properties are represented as green rectangles with their property name inside as seen in Figure 8.

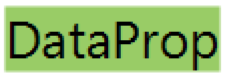


Figure 8.: Graph notation for data properties

3.1.4 *Annotation Properties*

Annotation Properties are represented as yellow rectangles with their property name inside as seen in Figure 9.

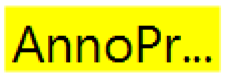


Figure 9.: Graph notation for annotation properties

3.1.5 *Datatypes*

Datatypes are represented as yellow rectangles with black borders that contain the name of their type inside as seen in Figure 10.

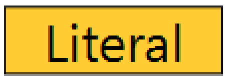


Figure 10.: Graph notation for datatypes

3.1.6 *Imported Classes and Properties*

Imported classes are represented as dark blue circles and imported properties are represented as dark blue rectangles as seen in Figures 11 and 12. This color overrides the default color of the element as seen in Figures 6 to 9.

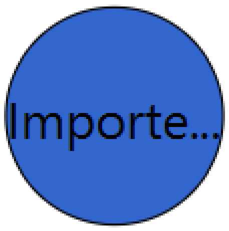


Figure 11.: Graph notation for imported classes

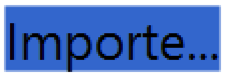


Figure 12.: Graph notation for imported properties

3.1.7 *Anonymous Elements*

Anonymous elements are represented with their default shape and colours but they do not contain any text inside as seen in Figures 13 and 14.

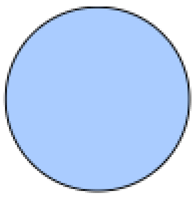


Figure 13.: Graph notation for anonymous classes

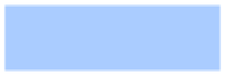


Figure 14.: Graph notation for anonymous object properties

3.1.8 *Element Characteristics*

Element characteristics (Functional, Inverse Functional, Transitive, Symmetric, Asymmetric, Reflexive and Irreflexive) are represented as white rectangles with a black border containing the name of the characteristic inside as seen in Figure 15.

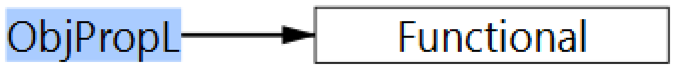


Figure 15.: Graph notation for a functional object property

3.1.9 *Other*

The remaining graph elements are used to connect several nodes together. They are represented by their element name and the lines connecting them to several other nodes. If the connection needs to connect to multiple nodes, an extra node (white colour, black border and a ∪ symbol) is added to the graph as seen in Figure 24.

Domain and Range

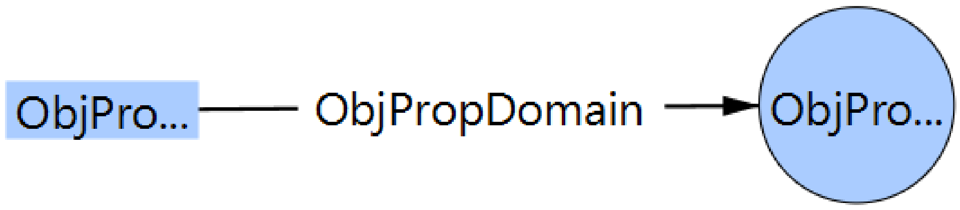


Figure 16.: Graph notation for the domain of an object property

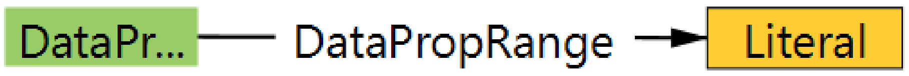


Figure 17.: Graph notation for the range of a data property

SubClass and SubProperty

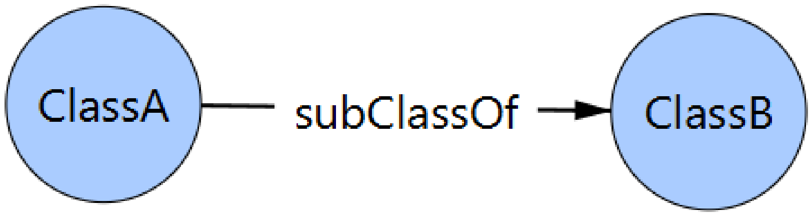


Figure 18.: Graph notation to display a subclass connection

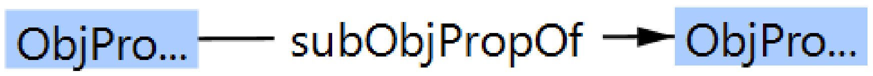


Figure 19.: Graph notation to display a sub object property connection

Equivalent

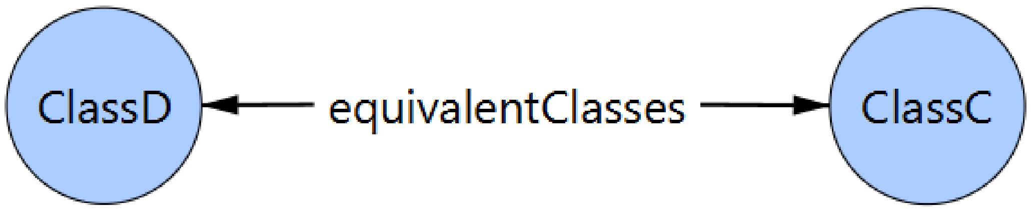


Figure 20.: Graph notation to display a equivalent class connection

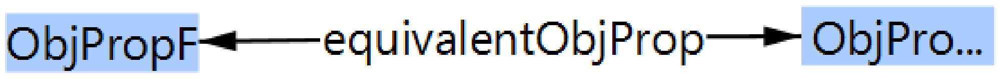


Figure 21.: Graph notation to display a equivalent object property connection

Inverse

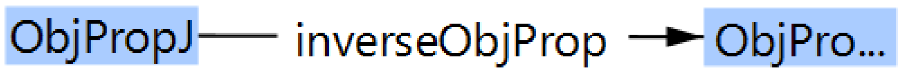


Figure 22.: Graph notation to display an inverse object property connection

Disjoint

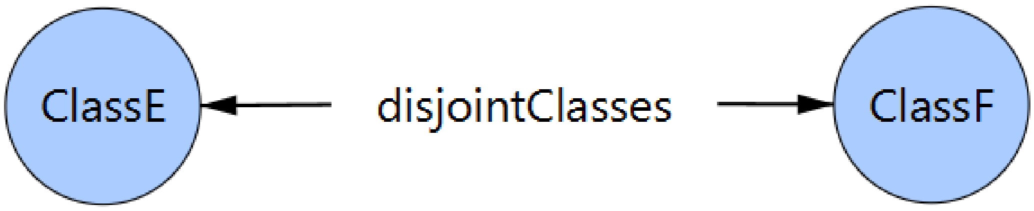


Figure 23.: Graph notation to display a disjoin classes connection

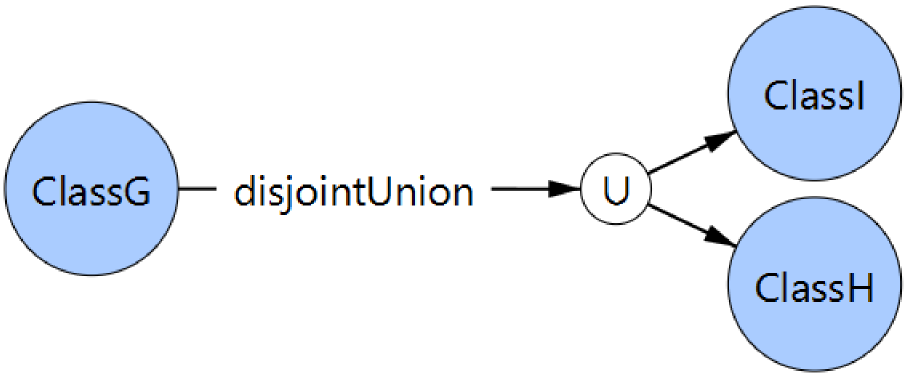


Figure 24.: Graph notation to display a disjoint union connection

3.1.10 *Difference Notation*

To display changes in the graph, the edges will be green or red and the nodes will display a border with a green or red colour. The green colour means that the element was added to the ontology and red colour means that the element was removed from the ontology.
In Figures 25 and 26 it is possible to observe the border that is created for circular and rectangle shaped nodes.

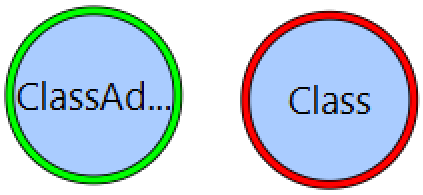


Figure 25.: Graph notation to display creation and removal of a class

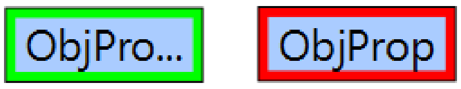


Figure 26.: Graph notation to display creation and removal of an object property

The removal of an axiom connecting multiple nodes can be seen in Figure 27.

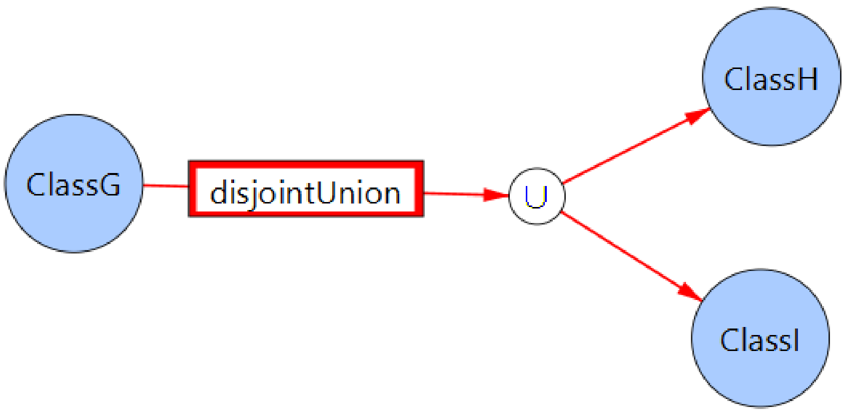


Figure 27.: Graph notation to display the removal of the *disjointUnion* axiom

It is possible to also have changes in the group of nodes targeted by an axiom. In Figure 28 it is possible to observe changes in the object property domain and the data property domain.

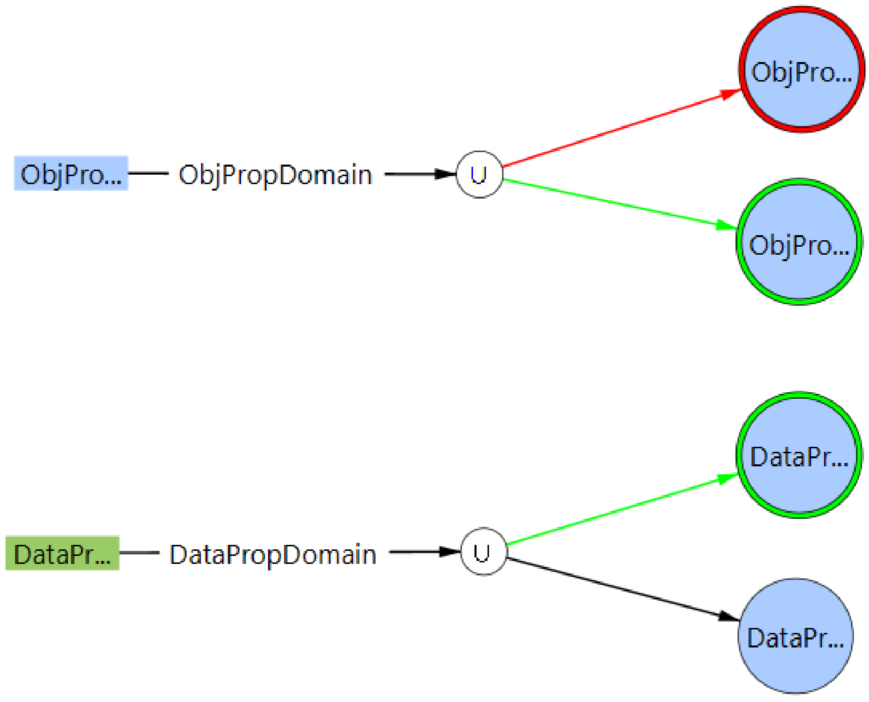


Figure 28.: Graph notation to display changes in domains

In Figure 28, concerning the object property domain we can see:

- Removal of a class to the ontology;
- Addition of a class to the ontology;

- The removed class in no longer part of the object property domain;
- The new class was added to the object property domain.

In Figure 28, concerning the data property domain we can see:

- Creation of a class to the ontology;
- The new class was added to the data property domain.

3.2 ONTODIFFGRAPH: ARCHITECTURE

Figure 29 depicts the architecture of the proposed OntoDiffGraph system. This system is able to load ontologies and display the differences between them through the use of a graph.

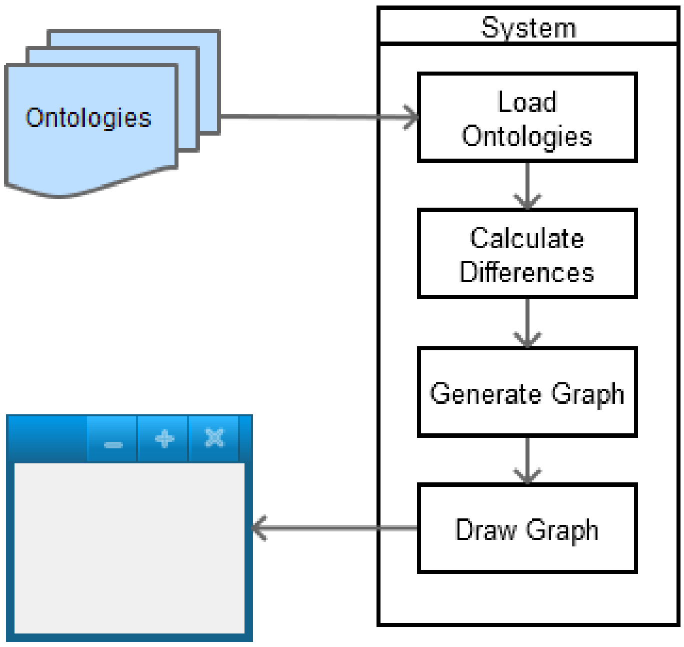


Figure 29.: Architecture of the OntoDiffGraph system

When using the system, in the first step the user selects two versions of an ontology file for OntoDiffGraph to load.
After this, the application calculates the differences between these versions and generates a graph with the elements and differences in the ontology. Before the transition into the final step the application adds the necessary information to the other existing elements of the user interface, such as the Axiom Diff Tab or the Ontology Elements Tab, that are described in Chapter 4.

In the final step, the application draws the graph so that the user is able to visualize and interact with it. Since the graph is not static, this step has to be repeated several times in order to redraw the positions of the elements when they are moved.

3.3 TECHNOLOGY CHOICE

3.3.1 *OWL API*

OWL API (Horridge and Bechhofer, 2011) is a Java library that implements the ontology language OWL. It allows us to load and query ontologies in order to obtain information about its elements, a feature that is essential to create our application.

3.3.2 *Java*

Since OWL API is a Java library, the language OntoDiffGraph was made in needs to be Java.

3.3.3 *JavaFX*

To be able to create the user interface there are two options in Java, Swing and JavaFX. We decided to chose JavaFX since it is the most modern library and we found it easier to use. The entire user interface including the graph is done using this library.

4

ONTODIFFGRAPH: DEVELOPMENT

This chapter contains detailed information about the features and decisions made in the development of the OntoDiffGraph application.

4.1 GRAPH INTERACTIVITY

For user friendly graph interactivity, some important features needed to be added to assure the minimal usability of the application. The following requirements were considered *must-have* and have been implemented in the application:

- Zooming, allows the users to zoom in and out so that the graph can be seen from different distances.
- Panning, allows the users to drag the camera.
- Dragging, allows the users to drag nodes of the graph to the desired position.

4.2 GRAPH LAYOUT

In order for the graph to adjust itself to a more suitable layout for visualization we implemented a force-directed algorithm.
For the algorithm to know where the nodes need to be positioned in each iteration, it needs to calculate the following variables:

- Vertex repulsion force (all vertices of the graph repel each other);
- Edge spring force, edges will simulate a spring where the force is stronger the further away the length of the edge is from the variable *springLength*;
- Attraction force to center of the graph, in order for the vertices to stay close to the center of the graph all of them have a small force attracting them to the position (0,0).

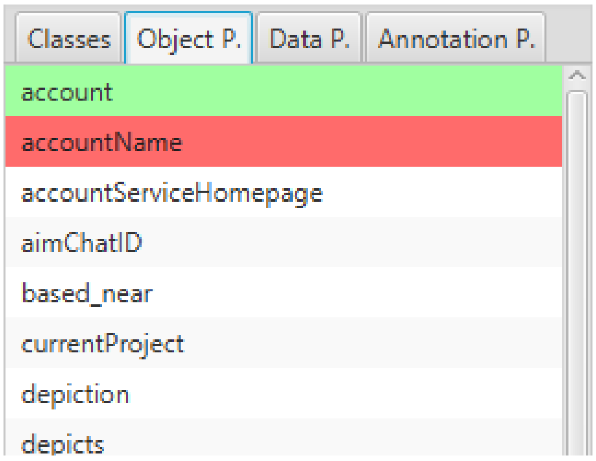
The algorithm that was implemented for the graph layout can be seen in the following pseudo-code:

```
Map<Vertex,Force> forces
deltaTime = timePassed * timestep

Foreach vertexA in VertexList {

   resultForceX = 0
   resultForceY = 0

   // Calculate repulsion force
   ForEach vertexB in VertexList {
      distBetweenVertices = max(distance(vertexA, vertexB), 1)
      repulsionForce = repulsionStrength / max(distanceBetweenVertices, 2)

      resultForceX += repulsionForce * (vertexA.x - vertexB.x) / distBetweenVertices
      resultForceY += repulsionForce * (vertexA.y - vertexB.y) / distBetweenVertices
   }

   // Calculate attraction force
   ForEach edge in EdgeList {
      edgeLength = max(edge.getLength(), 1)
      attractionForce = - attractionStrength * (edgeLength - springLength)

      resultForceX += attractionForce * edge.getXDistance() / edgeLength
      resultForceY += attractionForce * edge.getYDistance() / edgeLength
   }

   // Calculate force for attraction to center of graph
   resultForceX -= repulsionStrength / 100 000 000 * vertexA.x
   resultForceY -= repulsionStrength / 100 000 000 * vertexA.y

   forces.set(vertexA, new Force(resultForceX, resultForceY))
}

// Calculate force and move vertices
Foreach vertexA in VertexList {
   force = forces.get(vertexA)

   force.dampen(0.1);
   vertexA.move(force.x * deltaTime, force.y * deltaTime)
}
```

The default values for the variables: *repulsionForce, attractionForce, springLength, timestep,* can be seen in Figure 33.

4.3 ONTOLOGY ELEMENTS LIST

In OntoDiffGraph, it is also possible to easily find the location of an element in the graph by the use of the menu containing the list of classes, object properties, data properties and annotation properties of the ontology. By selecting an element of the list, the node in the visualization will be centered in the viewable area. This allows the user to easily locate the position of an element if they know its name, a feature that is helpful if the user is visualizing large ontologies. This menu is shown in the following figure:



account
accountName
accountServiceHomepage
aimChatID
based_near
currentProject
depiction
depicts

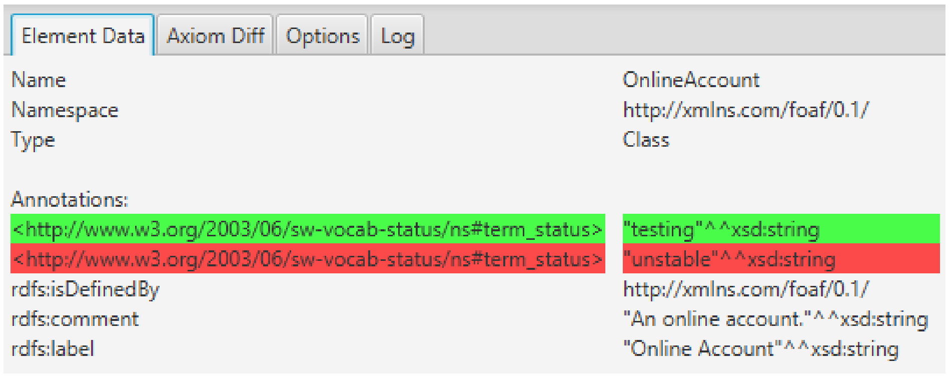
Figure 30.: Tabs with all ontology elements

In this menu it is also possible to see the elements of the ontology that were added or removed. The background colour of each cell shows what happened to that element between the different versions. The possible colours are:

- White, the element did not change from one version to the other;
- Green, the element was added to the ontology;
- Red, the element was removed from the ontology.

4.4 ELEMENT DATA

When a user clicks a graph node it is possible to see additional information about the node in the tab “Element Data”. In this tab the user can see the name, namespace, type and the annotations of the element corresponding to the node that was clicked on. However if the node is anonymous this tab will only display the textual version of this node.



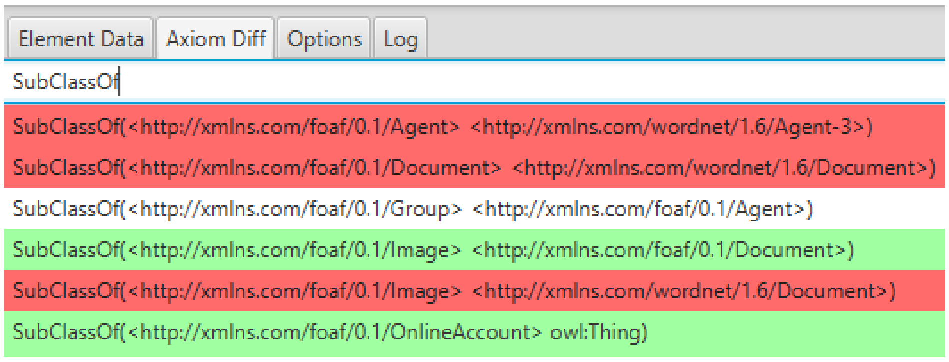
Element Data | Axiom Diff | Options | Log

Name	OnlineAccount
Namespace	http://xmlns.com/foaf/0.1/
Type	Class
Annotations:	
<http://www.w3.org/2003/06/sw-vocab-status/ns#term_status>	"testing"^^xsd:string
<http://www.w3.org/2003/06/sw-vocab-status/ns#term_status>	"unstable"^^xsd:string
rdfs:isDefinedBy	http://xmlns.com/foaf/0.1/
rdfs:comment	"An online account."^^xsd:string
rdfs:label	"Online Account"^^xsd:string

Figure 31.: Tab showing information about a node

4.5 AXIOM DIFFERENCE

In the axiom difference tab, it is possible to see all the axioms in both versions of the ontology. The elements of this list also contain the background colour green or red to show the user if the axiom was added or removed from the ontology. It is also possible to filter this list by using a text field located in the top of the list.
In Figure 32 it is possible to see all axioms of the type "SubClassOf" in the ontology and the changes that were made in these axioms.



Element Data | Axiom Diff | Options | Log

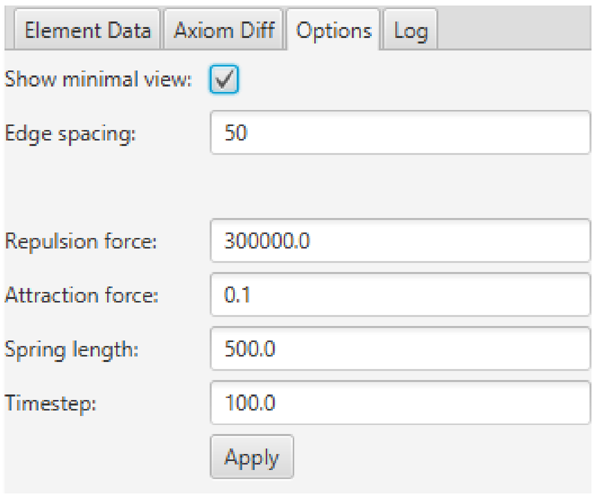
SubClassOf

```
SubClassOf(<http://xmlns.com/foaf/0.1/Agent> <http://xmlns.com/wordnet/1.6/Agent-3>)
SubClassOf(<http://xmlns.com/foaf/0.1/Document> <http://xmlns.com/wordnet/1.6/Document>)
SubClassOf(<http://xmlns.com/foaf/0.1/Group> <http://xmlns.com/foaf/0.1/Agent>)
SubClassOf(<http://xmlns.com/foaf/0.1/Image> <http://xmlns.com/foaf/0.1/Document>)
SubClassOf(<http://xmlns.com/foaf/0.1/Image> <http://xmlns.com/wordnet/1.6/Document>)
SubClassOf(<http://xmlns.com/foaf/0.1/OnlineAccount> owl:Thing)
```

Figure 32.: Tab showing the changes in axioms

4.6 GRAPH CUSTOMIZATION

In the application it is possible to change general and layout specific values during runtime. This makes the user able to easily learn the impact of each parameter in the graph visualization and is a better option than reloading the entire graph when changes are made on it.


Element Data | Axiom Diff | Options | Log

Show minimal view: ✓

Edge spacing: 50

Repulsion force: 300000.0

Attraction force: 0.1

Spring length: 500.0

Timestep: 100.0

Apply

Figure 33.: Layout options

In Figure 33 it is possible to visualize the currently available variables for the force-directed layout:

- Show minimal view, if enabled the graph will only show elements that are necessary for the user to visualize the difference in the ontologies. Elements that did not change will not be displayed unless they are related to the change, for example, if there is a change in the domain of a property all elements of that domain will be shown (even the ones that were not added or removed).
- Edge spacing, controls the distance between edges with the same node ends.
- Repulsion force, controls the repulsion force of the nodes of the graph.
- Attraction force, controls the attraction force of the edges of the graph.
- Spring length, controls the length of the edges.
- Timestep, controls the seconds that pass in every iteration of the algorithm.

4.7 CHANGE DETECTION

In order to find all the differences between two versions of an ontology it is necessary to compare their contents. Classes, object properties, data properties and annotation properties contain a unique IRI that identifies them. This helps identifying elements that were added or removed. The following list contains the changes made in an ontology and how it is possible to determine that change:

- Element was added: an element is classified as added to the ontology if its IRI does not exist in the initial version but exists in the final version;

- Element was removed: an element is classified as removed from the ontology if its IRI exists in the initial version but does not exist in the final version.

One weakness of this method is that renaming an element will change its IRI, therefore this change will be displayed as the removal of the element with the old name and the addition of a new element with the new name. This will cause all the edges connected to the old node to be removed and added as edges to the new node, making it seems that there were a lot of changes in the ontology when it was just a simple name change.

For the elements of the ontology that do not contain an IRI (subclass axioms, equivalent axioms, etc), the OWL API library will be used to find if the axiom was added or removed from the ontology.

4.8 VISUALIZATION

After the graph is created its elements will be drawn in a JavaFX Canvas node so that it can be displayed to the user. The graph uses a force-directed layout to organize the nodes and edges so that the user can more easily visualize the graph. This algorithm is running in a separate thread so that it does not impact the performance of the JavaFX application thread.
The graph created by this process in OntoDiffGraph can be seen in Figure 34, the image displays some elements of the FOAF ontology using a graph. In this image it is possible to observe the effect of the layout in the graph, the nodes repel each other while the edges work as springs, leading to a stretched-out and very clean layout that eases its visualization.

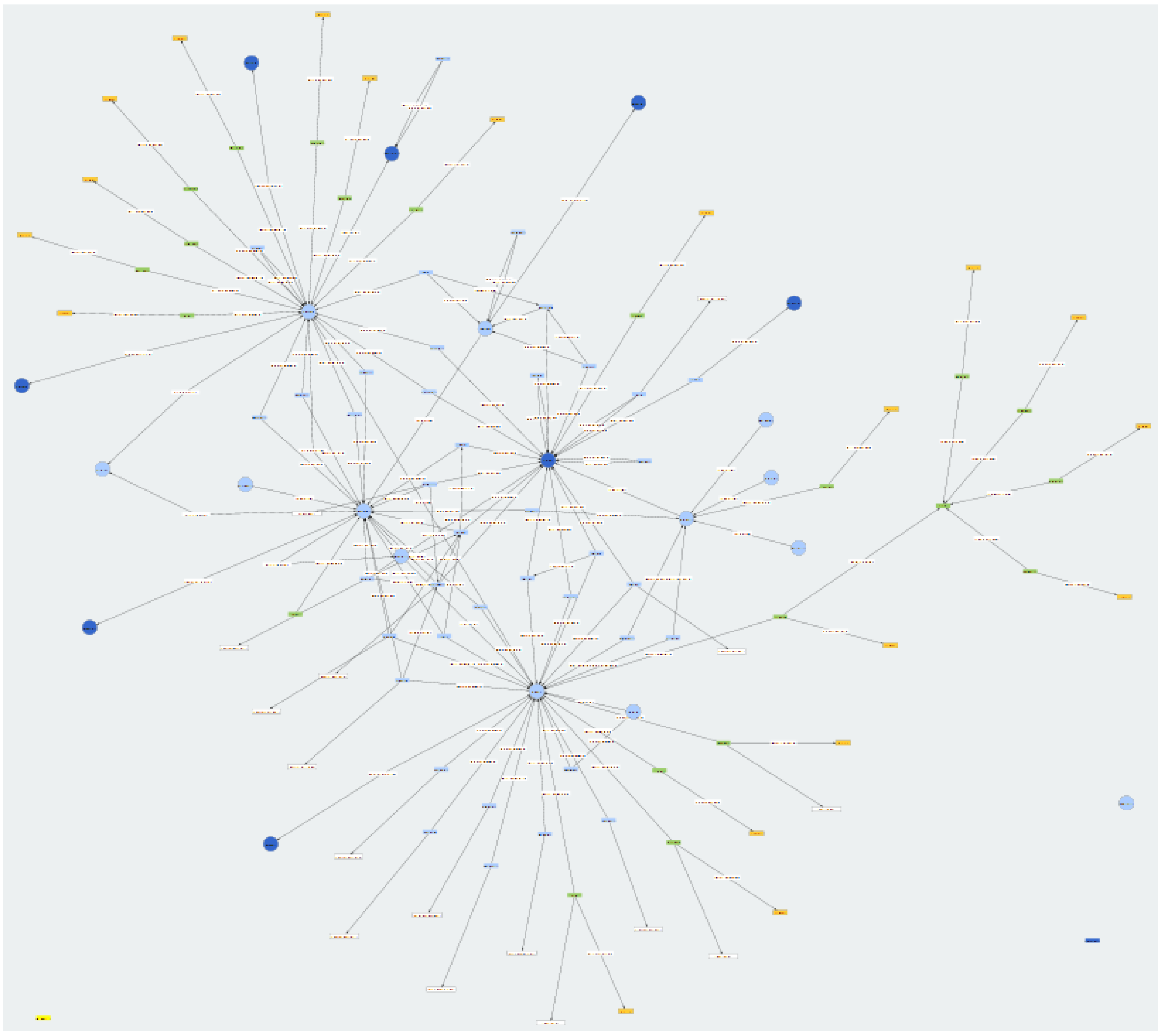


Figure 34.: FOAF ontology displayed in OntoDiffGraph

5

CASE STUDY

This case study was created to show how to visualize differences in an ontology using OntoDiffGraph. In this case study the MOD ontology (Dutta et al., 2015) was used to achieve this objective. The different versions of the ontology used in this case study are available online in `http://www.di.uminho.pt/~gepl/OntoDiffGraph/`.

We open OntoDiffGraph and click on the option "Create Ontology Diff" located in the "File" menu, as seen in Figure 35.

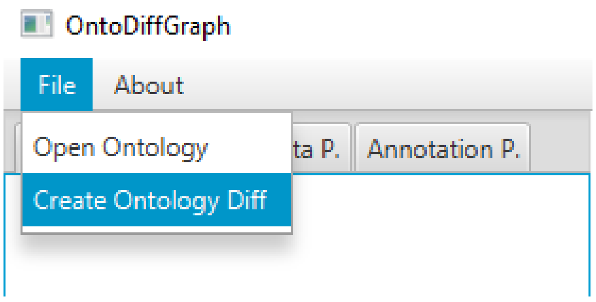
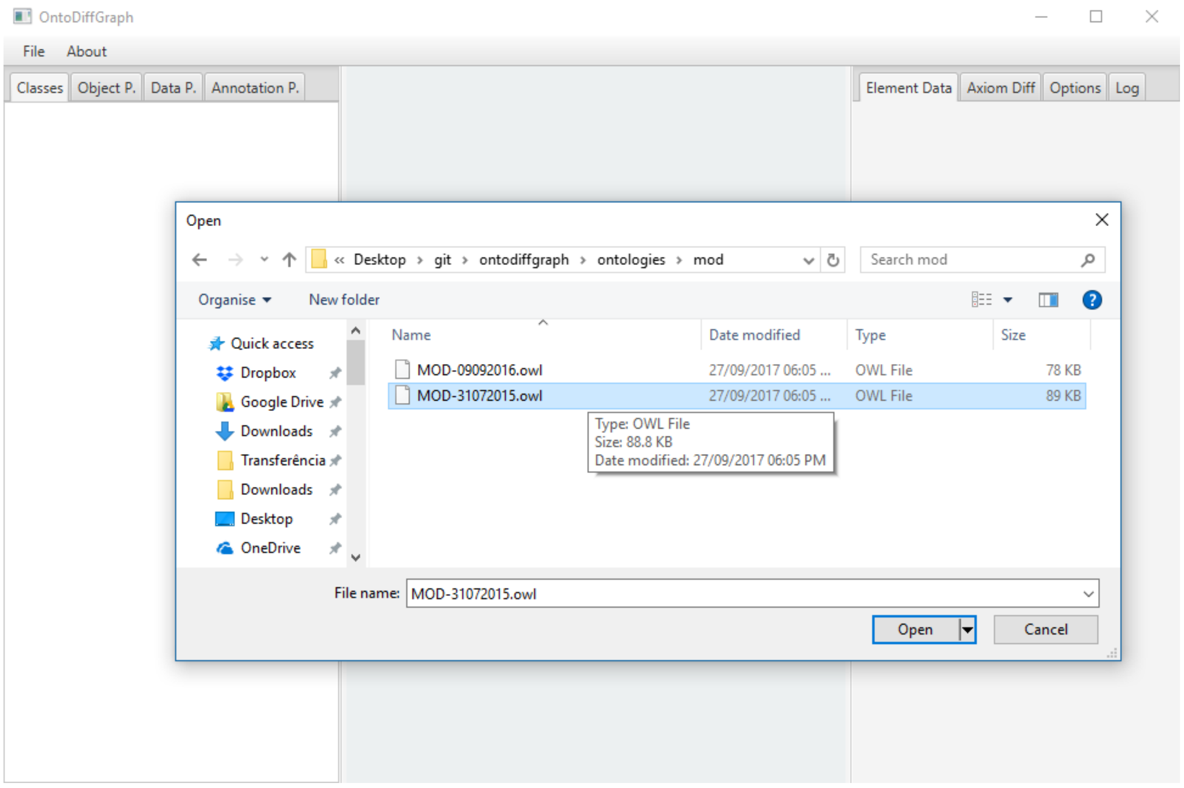


Figure 35.: Creating the ontology difference

After clicking the option, a window should pop out asking the user to select the older version of the ontology (Figure 36). We double click the older version of the MOD ontology and a new window should pop out.



Name	Date modified	Type	Size
MOD-09092016.owl	27/09/2017 06:05 ...	OWL File	78 KB
MOD-31072015.owl	27/09/2017 06:05 ...	OWL File	89 KB

Type: OWL File
Size: 88.8 KB
Date modified: 27/09/2017 06:05 PM

File name: MOD-31072015.owl

Figure 36.: Select older ontology version

The new window that pops out should ask for another file, the most recent version of the ontology. We double click the most recent version and OntoDiffGraph will start calculating the differences and generating the graph (Figure 37).

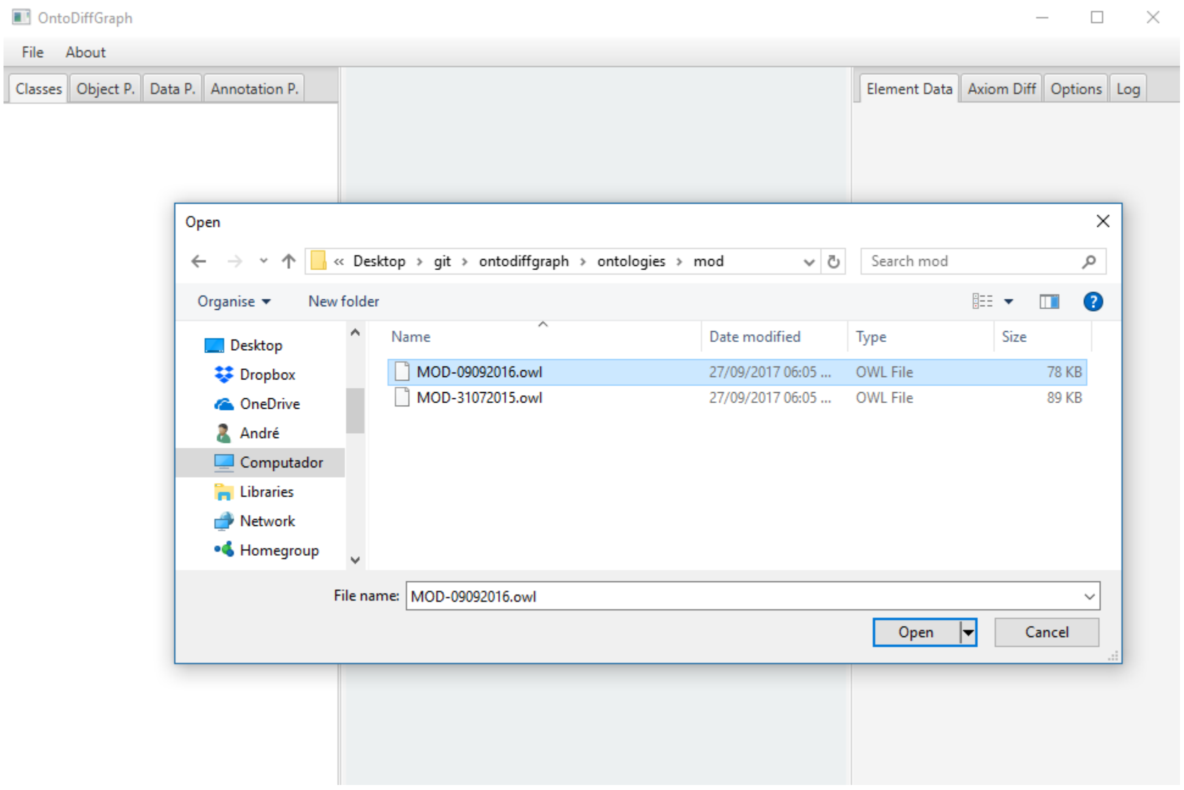


Figure 37.: Select most recent ontology version

At this step the graph should be visible to the user, however there is a lot of information that is not relevant in this case study, graph elements that don't display any difference can be hidden from the user by ticking the box "Show Minimal View" in the Options Tab (Figure 38).

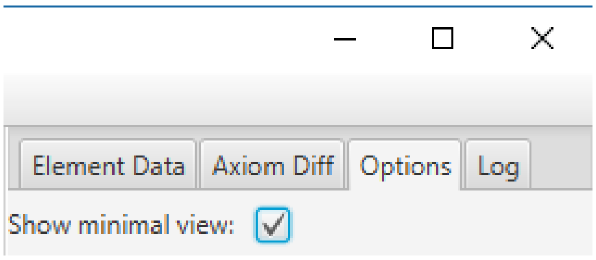
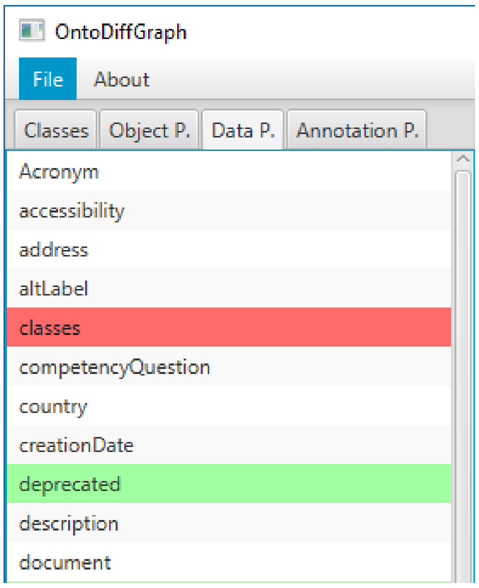


Figure 38.: Tick the "Show Minimal View" box

If the user wants to see what are the classes or object, data and annotation properties that were added/removed from the ontology, they can use the tabs located in the left side of the application (Figure 39). The elements are sorted by alphabetical order and have the colour green or red to display any change made to that element.


OntoDiffGraph

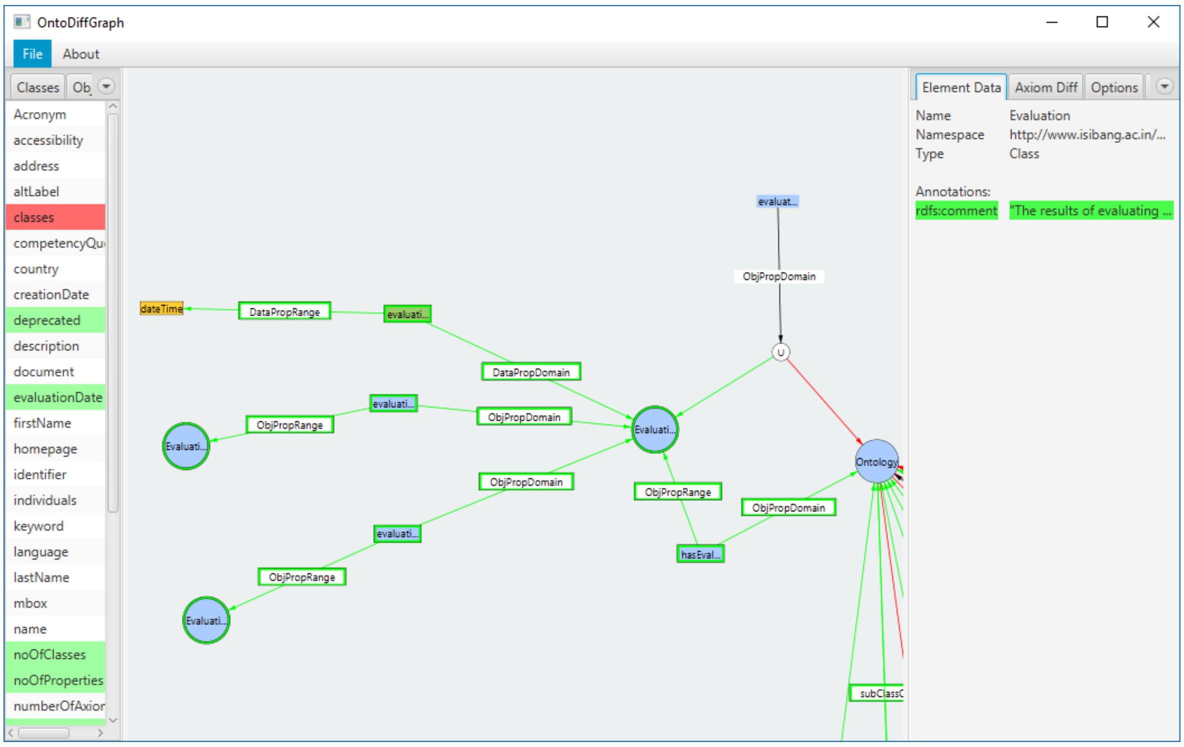
File About

Classes | Object P. | Data P. | Annotation P.

Acronym
accessibility
address
altLabel
classes
competencyQuestion
country
creationDate
deprecated
description
document

Figure 39.: Elements of the ontology

If the user wants instead to see the differences in the axioms related to an element, such as domains or ranges, the graph makes this process easier for the user (Figure 40). When there are problems locating an element in the graph the user can click an name from the list of elements in the left to quickly jump to a certain node in the graph.


OntoDiffGraph

File About

Classes | Ob

Acronym
accessibility
address
altLabel
classes
competencyQu
country
creationDate
deprecated
description
document
evaluationDate
firstName
homepage
identifier
individuals
keyword
language
lastName
mbox
name
noOfClasses
noOfProperties
numberOfAxior

Element Data | Axiom Diff | Options

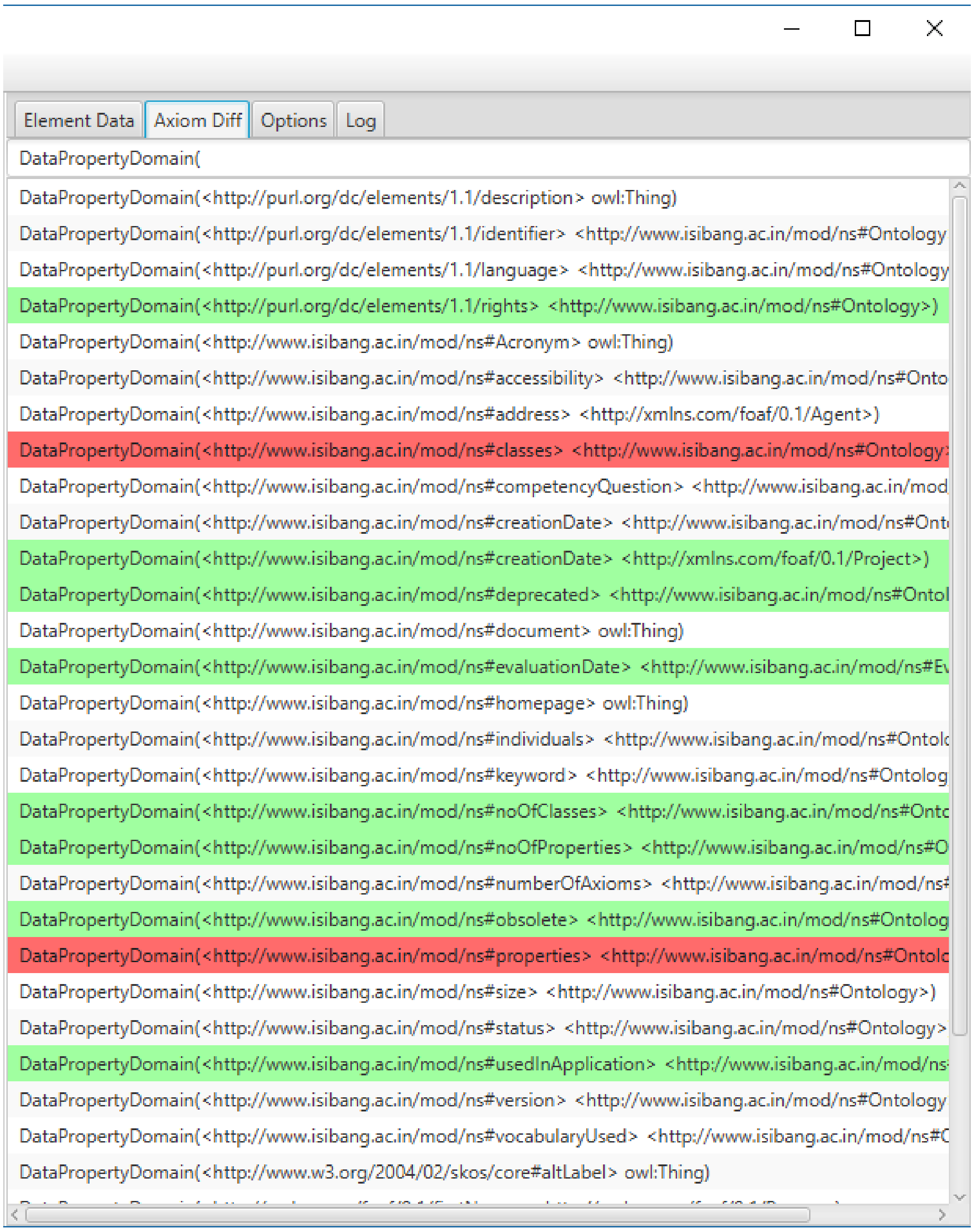
Name	Evaluation
Namespace	http://www.isibang.ac.in/...
Type	Class

Annotations:

rdfs:comment	"The results of evaluating ...

Figure 40.: Graph displaying some changes in the MOD ontology

However if the user is trying to find the answer to "How many DataPropertyDomain axioms were removed or added?" using the graph it will take a lot of time. This is because these axioms are scattered all over the graph, making it very hard for the user to find all of them. The best method to use in this scenario will be to use the tab "Axiom Diff" and filter the list of axioms as seen in Figure 41.



```
DataPropertyDomain(
DataPropertyDomain(<http://purl.org/dc/elements/1.1/description> owl:Thing)
DataPropertyDomain(<http://purl.org/dc/elements/1.1/identifier> <http://www.isibang.ac.in/mod/ns#Ontology
DataPropertyDomain(<http://purl.org/dc/elements/1.1/language> <http://www.isibang.ac.in/mod/ns#Ontology
DataPropertyDomain(<http://purl.org/dc/elements/1.1/rights> <http://www.isibang.ac.in/mod/ns#Ontology>)
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#Acronym> owl:Thing)
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#accessibility> <http://www.isibang.ac.in/mod/ns#Onto
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#address> <http://xmlns.com/foaf/0.1/Agent>)
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#classes> <http://www.isibang.ac.in/mod/ns#Ontology
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#competencyQuestion> <http://www.isibang.ac.in/mod
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#creationDate> <http://www.isibang.ac.in/mod/ns#Ont
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#creationDate> <http://xmlns.com/foaf/0.1/Project>)
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#deprecated> <http://www.isibang.ac.in/mod/ns#Ontol
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#document> owl:Thing)
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#evaluationDate> <http://www.isibang.ac.in/mod/ns#E
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#homepage> owl:Thing)
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#individuals> <http://www.isibang.ac.in/mod/ns#Ontol
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#keyword> <http://www.isibang.ac.in/mod/ns#Ontolog
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#noOfClasses> <http://www.isibang.ac.in/mod/ns#Ont
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#noOfProperties> <http://www.isibang.ac.in/mod/ns#O
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#numberOfAxioms> <http://www.isibang.ac.in/mod/ns
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#obsolete> <http://www.isibang.ac.in/mod/ns#Ontolog
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#properties> <http://www.isibang.ac.in/mod/ns#Ontol
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#size> <http://www.isibang.ac.in/mod/ns#Ontology>)
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#status> <http://www.isibang.ac.in/mod/ns#Ontology>
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#usedInApplication> <http://www.isibang.ac.in/mod/ns
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#version> <http://www.isibang.ac.in/mod/ns#Ontology
DataPropertyDomain(<http://www.isibang.ac.in/mod/ns#vocabularyUsed> <http://www.isibang.ac.in/mod/ns#
DataPropertyDomain(<http://www.w3.org/2004/02/skos/core#altLabel> owl:Thing)
```

Figure 41.: List of axioms of the ontology

Through the use of all the features available in OntoDiffGraph the user should be able to answer multiple types of questions about the differences in the versions of the MOD ontology.

6 EXPERIMENT

An experiment was made in order to evaluate the OntoDiffGraph application. The participants in this experiment had to use the developed application (OntoDiffGraph) and OWLDiff to discover the differences between versions of an ontology (FOAF (FOA) and MOD (Dutta et al., 2015)). In order to receive feedback, the participants had to answer questions about these differences and give their opinion about the existing functionalities of OntoDiffGraph and what could be improved on.
The questions the participants had to answer can be seen in Appendix A. This appendix contains:

- A guide on how to visualize the difference in versions of ontologies using OWLDiff and OntoDiffGraph;
- Several questions that the users needed to answer;
- The graph notation used by the OntoDiffGraph application.

6.1 EXPERIMENT SETUP

The experiment was done in an environment with two supervisors; the twelve participants used their own computers.
The participants' age range is between 25 and 35 years old and all of them were from the field of computer science and had previous experience with ontologies. Three of the participants have a PhD, six were still completing their PhD and the other three were doing their master's. They did not have access to OntoDiffGraph and were not informed that OWLDiff was going to be used before the experiment. In order to avoid problems with the setup all participants were previously requested to download a virtual machine that contained all the necessary tools to successfully complete the experiment. The disk of the virtual machine was encrypted and the decryption key was only given to the participants at the start of the experiment, so that all participants started with an equivalent level of experience with the tools.

The experiment lasted for 1 hour and 30 minutes and all participants were able to complete it before the time limit.

6.2 RESULTS

In this section it is possible to observe the results of the experiment. Sections 6.2.1 and 6.2.1 contain questions about the FOAF and MOD ontologies respectively, both of them contain six questions and the tool that the participant must use to solve it is displayed at the start of the question. The first question of the OWLDiff application is similar to the first question of the OntoDiffGraph application and this pattern is repeated in the second and third questions. This was made on purpose so that it was possible to compare the results of each pair of questions.
Below the questions made to the participants we display the number of participants that answered correctly to the question.
In Section 6.2.3 the participants were asked to rate the applications that were used, their skills and to share their opinion about several statements. In the last question the participants were able to write about features that they thought OntoDiffGraph was lacking. Using these answers we were able to obtain helpful feedback on how to improve the application. The ones that were considered the most relevant answers are shown at the end of this section.

6.2.1 *FOAF Ontology*

1. [Protégé and OWLDiff] What is the object property that was added to the ontology?
   All 12 participants answered correctly to this question.
2. [Protégé and OWLDiff] How many data properties were added that have the range *Literal*?
   All 12 participants answered correctly to this question.
3. [Protégé and OWLDiff] What is the name of the external class (imported from another ontology) that was added to the FOAF ontology?
   8 out of the 12 participants answered this question correctly.
4. [OntoDiffGraph] What is the object property that was removed from the ontology?
   All 12 participants answered correctly to this question.
5. [OntoDiffGraph] What is the name of the functional data property that was added to the ontology?
   All 12 participants answered correctly to this question.

6. [OntoDiffGraph] What are the names of the external classes (imported from another ontology) removed from the FOAF ontology?
   All 12 participants answered correctly to this question.

6.2.2 *MOD Ontology*

1. [Protégé and OWLDiff] What happened to the domain of the data property *creationDate* ?
   8 out of 12 participants answered this question correctly.
2. [Protégé and OWLDiff] How many *subClassOf* axioms were added from the class *Ontology*?
   10 out of 12 participants answered this question correctly.
3. [Protégé and OWLDiff] The class *Person* was added to the range of which property?
   11 out of 12 participants answered this question correctly.
4. [OntoDiffGraph] What happened to the domain of the object property *evaluatedBy*?
   6 out of 12 participants answered this question correctly.
5. [OntoDiffGraph] How many *subClassOf* axioms were removed from the class *Ontology*?
   11 out of 12 participants answered this question correctly.
6. [OntoDiffGraph] The class *Project* was added to the domain of which data property?
   10 out of 12 participants answered this question correctly.

6.2.3 *Feedback*

- How would you rate the following tools in a scale from 1 - 10?
  - OWLDiff: 4.92 (Average)
  - OntoDiffGraph 7.75 (Average)
- How would you rate the your proficiency using Protégé in a scale from 1 - 10?
  5.25 (Average)
- Have you worked with the OWLDiff plugin before taking this comparative evaluation?
  None of the users worked with OWLDiff before the experiment
- To the following statement "The graph helps the user visualize the difference between versions of the ontology." the participants chose:

- Strongly Agree: 6 out of 12 chose this option.
- Agree: 5 out of 12 chose this option.
- Neutral: No one chose this option.
- Disagree: 1 out of 12 chose this option.
- Strongly Disagree: No one chose this option.

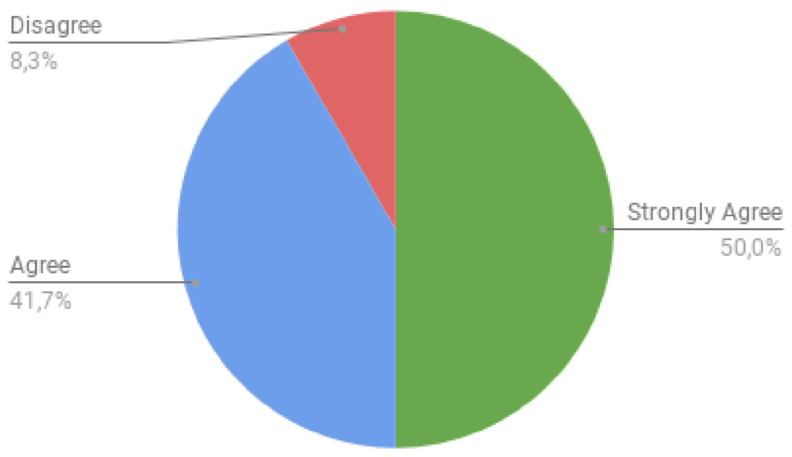


Figure 42.: Participant responses for the statement

- To the following statement "The graph notation is easy to understand." the participants chose:
  - Strongly Agree: 4 out of 12 chose this option.
  - Agree: 7 out of 12 chose this option.
  - Neutral: 1 out of 12 chose this option.
  - Disagree: No one chose this option.
  - Strongly Disagree: No one chose this option.

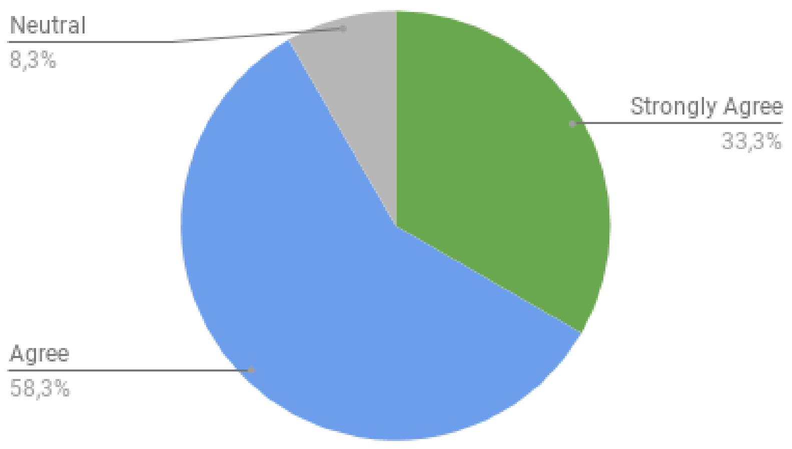


Figure 43.: Participant responses for the statement

- To the following statement "OWLDiff users also need to rely on Protégé to be able to understand some of the information displayed in the plugin interface." the participants chose:
  - Strongly Agree: 2 out of 12 chose this option.
  - Agree: 5 out of 12 chose this option.
  - Neutral: 3 out of 12 chose this option.
  - Disagree: 2 out of 12 chose this option.
  - Strongly Disagree: No one chose this option.

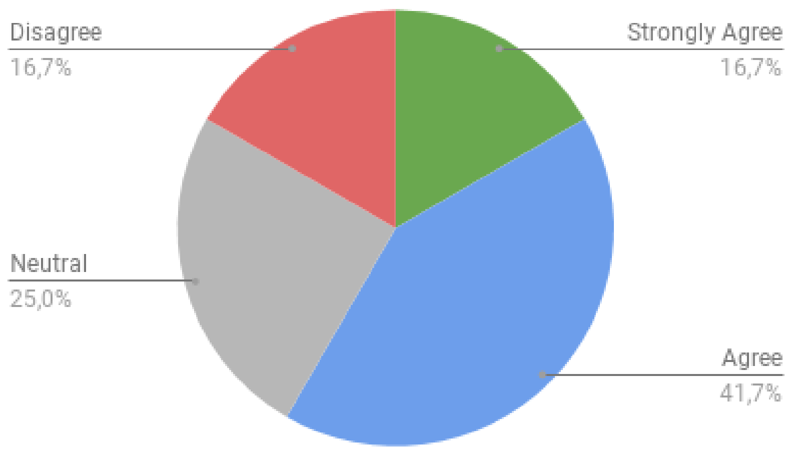


Figure 44.: Participant responses for the statement

A lot of the participants gave helpful feedback in the last question (Are there features that you think OntoDiffGraph is lacking? In this case, which ones?). The feedback that was considered most relevant is the following:

- When an element is selected in the Ontology Elements list the "Element Data" tab should also display its contents.
- Better filters for the Axiom Diff Tab.
- Add a filter for graph elements.
- Easier access to changes made in axioms related to an element.

6.3 DISCUSSION

The experiment was a success, there were no problems with the setup and all participants were able to answer all the questions on time.
The results of the experiment were also very positive, in the first ontology (FOAF) 4 out of the 12 participants had problems finding an external class that was added to the ontology. This is probably due to the fact that OWLDiff does not show the full URI of the element in the results and the participants had to rely on Protégé to find this information.
In the second ontology the participants made a lot more errors, from 4 wrongs answers in the FOAF ontology to 16 wrong answers in the MOD ontology. A lot of these errors are probably due to an increase of difficulty of the questions. It's interesting to note the high number of wrong answers that were made to the question about the changes made to a domain in the MOD ontology using OntoDiffGraph, which means users had difficulties answering these questions. This might be because to display a domain axiom that contains changes, the application has multiple graph elements. The domain edge is connected to a union node that links together all the nodes that are part of the domain and these edges can have different colours, thus, the participants might have needed more time to properly understand these types of changes.

The results of the tests show that both applications had a number of wrong answers close to each other, OWLDiff had 11 wrong answers and OntoDiffGraph had 9 wrong answers. However it is possible to observe that the users thought that OntoDiffGraph is a better tool than OWLDiff, and that the graph helps the user visualize differences easier and that its notation is easy to understand.
A somewhat controversial statement was the dependency of OWLDiff on Protégé. A lot of users did not agree with this statement, however some of the previous questions in OWLDiff required the use of Protégé to be answered correctly.

With this experiment it was possible to obtain information about the performance of OntoDiffGraph when compared with OWLDiff. The participants were able to perform at a similar level with both of these tools, however most of them rated OntoDiffGraph higher then OWLDiff and at the end of the experiment they expressed the opinion that OntoDiffGraph was easier to use and their preferred tool to use. The results of this experiment align with what we were expecting in the research hypothesis. The participants found OntoDiffGraph easier to use and helpful at locating certain differences when compared to OWLDiff.

7 CONCLUSION

This dissertation proposes an new technique to ease the visualization of the difference between two versions of an ontology.
Before starting the development of this technique it was necessary to research the domain of this dissertation. Research was done on the topic of ontology visualization, change detection in ontologies and the visualization of these changes.
Through this it was possible to obtain valuable information about already existing applications and notations that display ontology information through the use of several different techniques, algorithms used to detect ontology differences and applications that allow the user to visualize the difference between ontologies. The most important discovery in this research was the VOWL notation. The notation that was defined for OntoDiffGraph extends VOWL in order to also display information about the differences between versions of an ontology.
Through the use of the developed technique, it was possible to observe the differences by using a graph with a specialized notation, which proves the research hypothesis that setup this master's project.
In a small experiment with twelve participants it was possible to observe a preference for OntoDiffGraph over OWLDiff. In fact, most participants agreed that the use of a graph helps users visualize the difference between two versions of an ontology. The participants also had a similar amount of correct and wrong questions for the OntoDiffGraph and OWLDiff, showing that the use of this technique did not have a negative impact in the performance of the users at locating the differences. Some of the participants in the experiment used the Axiom Diff Tab over the graph to search for some of the differences, showing that it might be beneficial to have several methods for displaying the differences. This way, the user can choose the one that is the most adequate to perform the task.

7.1 CONTRIBUTIONS

In this work we show that the use of graphs to display ontologies can be adapted to also include additional information, for instance to understand the changes made on a version

of an ontology, and this transformation is preferred when compared to a less interactive or more cumbersome method of visualization.
It was also defined a notation which extends the already existing VOWL notation with the addition of elements that allow its users to visualize the difference between ontologies.
The OntoDiffGraph application was developed, enabling its users to see the difference between versions of an ontology, the detection of this difference being done using an algorithm that was also developed during the implementation of the application. This application can be obtained in http://www.di.uminho.pt/~gepl/OntoDiffGraph/, however the user must make sure Java is already installed in the system for it to execute.
To prove OntoDiffGraph was able to perform at the same level or above of already existing solutions we made an experiment whose execution plan could be replicated for other experiments.
An article was also already published for the SLATE 2017 conference (Lara et al., 2017) during the development of the application.

7.2 FUTURE WORK

Despite the amount of features already provided by OntoDiffGraph there is also a lot of improvements that can be made. Participants in the experiment gave very important feedback on what could be added or changed in the application.
Users requested that when an element is selected in the Ontology Elements list the "Element Data" tab should also display its contents — this was added to the application. However the other points raised by the users were not yet done and are left for future work.
The addition of more filters to the Axiom Diff Tab would help the users locate certain axioms easier, some of the filters that can be implemented are filtering by axiom type or by an IRI contained in the axiom.
Another filter that was requested is for the graph itself, to make the user able to hide graph elements that are not relevant.

For further future work, a Protégé plugin or a website with the functionality of OntoDiffGraph could be developed, giving users multiple ways (through a plugin or a browser) of trying the functionalities implemented in the application that was developed.

BIBLIOGRAPHY

Foaf vocabulary specification. http://xmlns.com/foaf/spec/. Accessed: 2016-09-23.

Harith Alani. Tgviztab: An ontology visualisation extension for protégé. Event Dates: October 26, 2003. URL http://eprints.soton.ac.uk/258326/.

Benjamin Bach, Emmanuel Pietriga, Ilaria Liccardi, and Gennady Legostaev. Ontotrix: a hybrid visualization for populated ontologies. In *Proceedings of the 20th international conference companion on World wide web*, pages 177–180. ACM, 2011.

Janis Barzdins, Guntis Barzdins, Karlis Cerans, Renars Liepins, and Arturs Sprogis. Owlgred: a uml style graphical notation and editor for owl 2. In *OWLED*, 2010.

Dave Beckett. RDF/xml syntax specification (revised). W3C recommendation, W3C, February 2004. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

Alessio Bosca, Dario Bonino, and Paolo Pellegrino. Ontosphere: more than a 3d ontology visualization tool. In *Swap*. Citeseer, 2005.

Mike Dean and Guus Schreiber. OWL web ontology language reference. W3C recommendation, W3C, February 2004. http://www.w3.org/TR/2004/REC-owl-ref-20040210/.

Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry*, 4(5):235–282, 1994.

Biswanath Dutta, Durgesh Nandini, and Gautam Kishore Shahi. Mod: Metadata for ontology description and publication. In *International Conference on Dublin Core and Metadata Applications*, pages 1–9, 2015.

Ramanathan Guha and Dan Brickley. RDF schema 1.1. W3C recommendation, W3C, February 2014. http://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

Florian Haag, Steffen Lohmann, Stefan Negru, and Thomas Ertl. Ontovibe: An ontology visualization benchmark. In *VISUAL@ EKAW*. Citeseer, 2014.

Florian Haag, Steffen Lohmann, Stephan Siek, and Thomas Ertl. Visual querying of linked data with queryvowl. *Joint Proceedings of SumPre*, 2015.

Michael Hartung, Anika Gross, and Erhard Rahm. Codex: exploration of semantic changes between ontology versions. *Bioinformatics*, 2012.

Michael Hartung, Anika Groß, and Erhard Rahm. Conto–diff: generation of complex evolution mappings for life science ontologies. *Journal of biomedical informatics*, 46(1):15–32, 2013.

Nathalie Henry, Jean-Daniel Fekete, and Michael J McGuffin. Nodetrix: a hybrid visualization of social networks. *IEEE transactions on visualization and computer graphics*, 13(6): 1302–1309, 2007.

Matthew Horridge and Sean Bechhofer. The owl api: A java api for owl ontologies. *Semantic Web*, 2(1):11–21, 2011.

Ajaz Hussain, Khalid Latif, Aimal Tariq Rextin, Amir Hayat, and Masoon Alam. Scalable visualization of semantic nets using power-law graphs. *Applied Mathematics & Information Sciences*, 8(1):355, 2014.

Boris Konev, Dirk Walther, and Frank Wolter. The logical difference problem for description logic terminologies. In *International Joint Conference on Automated Reasoning*, pages 259–274. Springer, 2008.

Petr Kremen, Marek Smid, and Zdenek Kouba. Owldiff: A practical tool for comparison and merge of owl ontologies. In *2011 22nd International Workshop on Database and Expert Systems Applications*, pages 229–233. IEEE, 2011.

Simone Kriglstein and Günter Wallner. Knoocks-a visualization approach for owl lite ontologies. In *Complex, Intelligent and Software Intensive Systems (CISIS), 2010 International Conference on*, pages 950–955. IEEE, 2010.

André Lara, Pedro Rangel Henriques, and Alda Lopes Gançarski. Visualization of ontology evolution using ontodiffgraph. In *OASIcs-OpenAccess Series in Informatics*, volume 56. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

Vincent Link, Steffen Lohmann, and Florian Haag. OntoBench: Generating custom OWL 2 benchmark ontologies. In *Proceedings of the 15th International Semantic Web Conference (ISWC '16)*, volume 9982 of *Lecture Notes in Computer Science*, pages 122–130. Springer, 2016.

Steffen Lohmann, Vincent Link, Eduard Marbach, and Stefan Negru. Webvowl: Web-based visualization of ontologies. In *International Conference on Knowledge Engineering and Knowledge Management*. Springer, 2014a.

Steffen Lohmann, Stefan Negru, and David Bold. The protégévowl plugin: ontology visualization for everyone. In *European Semantic Web Conference*. Springer, 2014b.

Steffen Lohmann, Stefan Negru, Florian Haag, and Thomas Ertl. Vowl 2: user-oriented visualization of ontologies. In *International Conference on Knowledge Engineering and Knowledge Management*. Springer, 2014c.

Enrico Motta, Paul Mulholland, Silvio Peroni, Mathieu d'Aquin, Jose Manuel Gomez-Perez, Victor Mendez, and Fouad Zablith. A novel approach to visualizing and navigating ontologies. In *International Semantic Web Conference*, pages 470–486. Springer, 2011.

Natalya Fridman Noy, Mark A Musen, et al. Promptdiff: A fixed-point algorithm for comparing ontology versions. *AAAI/IAAI*, 2002.

Bijan Parsia, Taowei Wang, and Jennifer Golbeck. Visualizing web ontologies with cropcircles. In *Proceedings of the 4th International Semantic Web Conference*, pages 6–10. Citeseer, 2005.

David Stephen John Perrin. *Prompt-viz: Ontology version comparison visualizations with treemaps*. PhD thesis, Citeseer, 2004.

Miguel Ángel Rodrıguez-Garcıa, Luke Slater, Keiron O'Shea, Paul N Schofield, Georgios V Gkoutos, and Robert Hoehndorf. Visualizing ontologies with aberowl.

Margaret-Anne Storey, Mark Musen, John Silva, Casey Best, Neil Ernst, Ray Fergerson, and Natasha Noy. Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in protégé. In *Workshop on interactive tools for knowledge capture*, pages 417–423, 2001.

A

DOCUMENTATION TO SUPPORT THE EXPERIMENT

This chapter contains the documentation that was used in the experiment. This chapter is divided in two sections, Appendices A.1 and A.2, the former has an explanation on how to visualize differences in OntoDiffGraph and OWLDiff and the latter has the examples that were shown to the participants and the questions they had to answer.
There was also another section in the documentation used in the experiment, the OntoDiff-Graph graph notation, however this is not in Appendix A since this information is already displayed in Section 3.1.

A.1 ONTODIFFGRAPH AND OWLDIFF GUIDE

A.1. OntoDiffGraph and OWLDiff guide

INTRODUCTION

Thanks for participating in the evaluation of the OntoDiffGraph application.
This application was created to allow the user to easily identify changes made in an ontology. In order to achieve this, the application displays the ontology in a graph with a well defined notation and contains a user interface with useful tools.
The objective of this questionnaire is to obtain information about the use of this application by real users and to compare its performance to an already existing solution (OWLDiff).

TOOLS

OWLDiff

OWLDiff is a plugin used by Protégé, a free and open-source ontology editor. Protégé can be downloaded from `https://protege.stanford.edu/`. If OWLDiff is already installed in Protégé, the user should be able to access the "Compare Ontologies..." feature as seen in Figure 45.

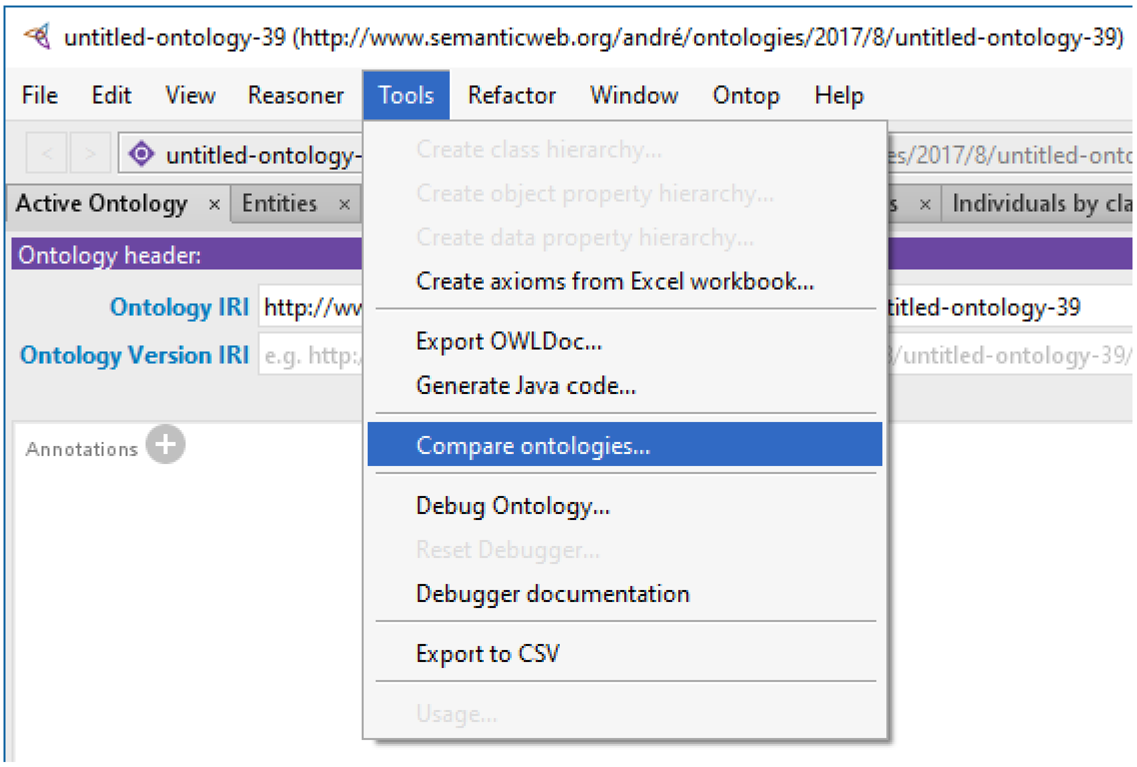


Figure 45.: Compare Ontologies feature

If this option is not visible, this means that the plugin is not installed. To install OWLDiff the user needs to search for existing plugins as shown in Figure 46, and then install the plugin "OWL Difference" seen in Figure 47.

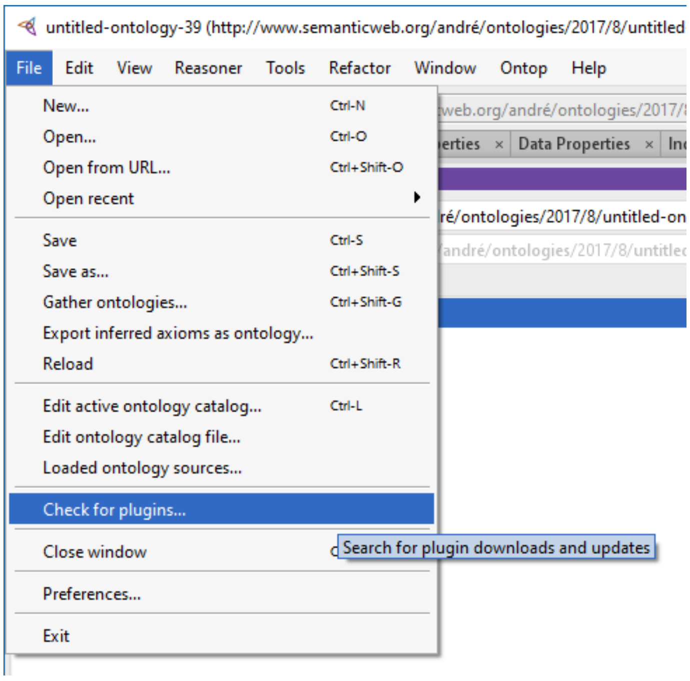
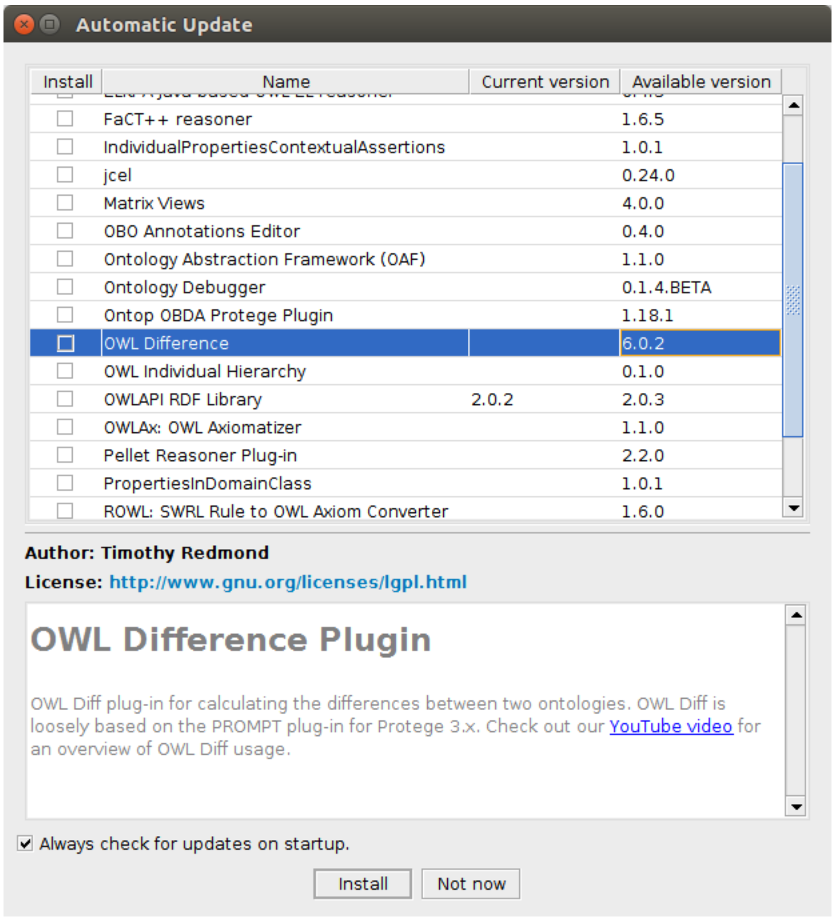

untitled-ontology-39 (http://www.semanticweb.org/andré/ontologies/2017/8/untitled
File
Edit
View
Reasoner
Tools
Refactor
Window
Ontop
Help
New... Ctrl-N
Open... Ctrl-O
Open from URL... Ctrl+Shift-O
Open recent
Save Ctrl-S
Save as... Ctrl+Shift-S
Gather ontologies... Ctrl+Shift-G
Export inferred axioms as ontology...
Reload Ctrl+Shift-R
Edit active ontology catalog... Ctrl-L
Edit ontology catalog file...
Loaded ontology sources...
Check for plugins...
Close window
Preferences...
Exit
Search for plugin downloads and updates
Data Properties

Figure 46.: Check for existing plugins



Automatic Update

Install	Name	Current version	Available version
☐	FaCT++ reasoner		1.6.5
☐	IndividualPropertiesContextualAssertions		1.0.1
☐	jcel		0.24.0
☐	Matrix Views		4.0.0
☐	OBO Annotations Editor		0.4.0
☐	Ontology Abstraction Framework (OAF)		1.1.0
☐	Ontology Debugger		0.1.4.BETA
☐	Ontop OBDA Protege Plugin		1.18.1
☐	OWL Difference		6.0.2
☐	OWL Individual Hierarchy		0.1.0
☐	OWLAPI RDF Library	2.0.2	2.0.3
☐	OWLAx: OWL Axiomatizer		1.1.0
☐	Pellet Reasoner Plug-in		2.2.0
☐	PropertiesInDomainClass		1.0.1
☐	ROWL: SWRL Rule to OWL Axiom Converter		1.6.0

Author: Timothy Redmond
License: http://www.gnu.org/licenses/lgpl.html

OWL Difference Plugin

OWL Diff plug-in for calculating the differences between two ontologies. OWL Diff is loosely based on the PROMPT plug-in for Protege 3.x. Check out our YouTube video for an overview of OWL Diff usage.

☑ Always check for updates on startup.

Install | Not now

Figure 47.: OWL Difference Plugin

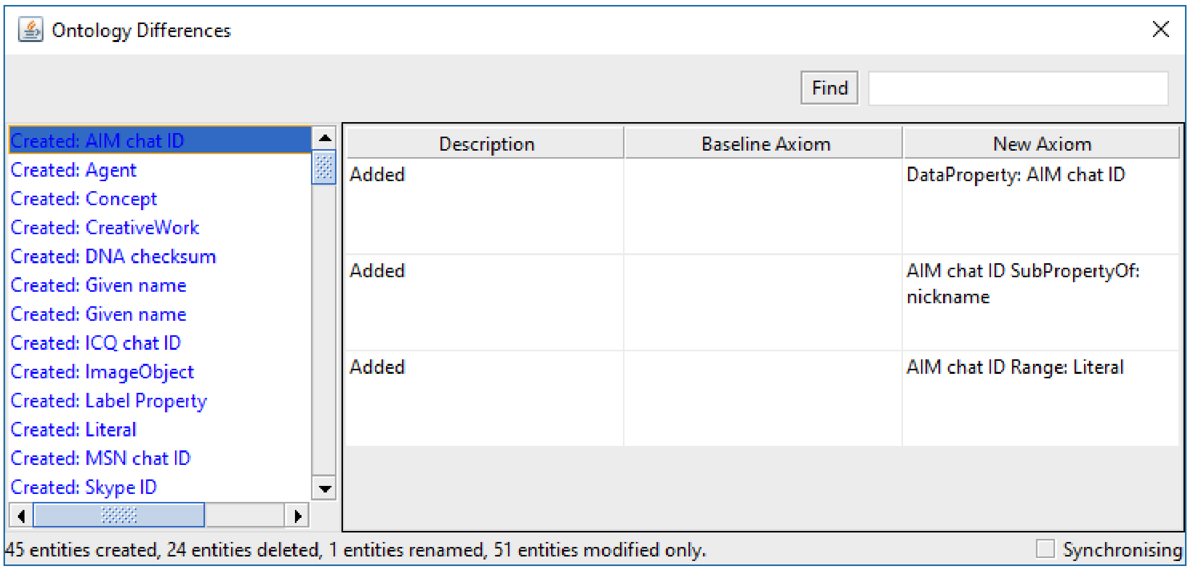
Displaying the difference

To display the difference between two ontologies the user needs to:

- Open the most recent ontology version (File > Open...)
- Open the plugin window (Tools > Compare Ontologies...)
- Inside the plugin window, "Browse" and select the older version of the ontology
- Enable the option "Open original ontology in separate workspace"
- Press OK to open the plugin window showing the differences

Detecting differences

The window in Figure 48 displays some of the differences in the FOAF ontology.


Ontology Differences

Find

Created: AIM chat ID
Created: Agent
Created: Concept
Created: CreativeWork
Created: DNA checksum
Created: Given name
Created: Given name
Created: ICQ chat ID
Created: ImageObject
Created: Label Property
Created: Literal
Created: MSN chat ID
Created: Skype ID

Description	Baseline Axiom	New Axiom
Added		DataProperty: AIM chat ID
Added		AIM chat ID SubPropertyOf: nickname
Added		AIM chat ID Range: Literal

45 entities created, 24 entities deleted, 1 entities renamed, 51 entities modified only.

Figure 48.: Displaying differences in FOAF ontology

In this picture it's possible to observe that the entity "Created: AIM chat ID" is selected. Selecting an entity in that list will show the axioms that changed in the table to the right. In this list an entity can appear multiple times if it contains more than one type of change, for example:

- Created: AIM Chat ID
- Deleted: AIM Chat ID (Not visible in image)

The user needs to click an element in order to find the changes made to its axioms. The table that is displayed when a user clicks an entity contains three columns:

- Description, the type of change
- Baseline Axiom, the axiom that exists in the original ontology
- New Axiom, the axiom that exists in the most recent ontology

To fully understand some of the changes displayed on the plugin the user might need to search the entities in Protégé. This can happen if there are multiple entities with the same name but from different ontologies, the OWLDiff plugin does not show the full IRI of an entity.

OntoDiffGraph

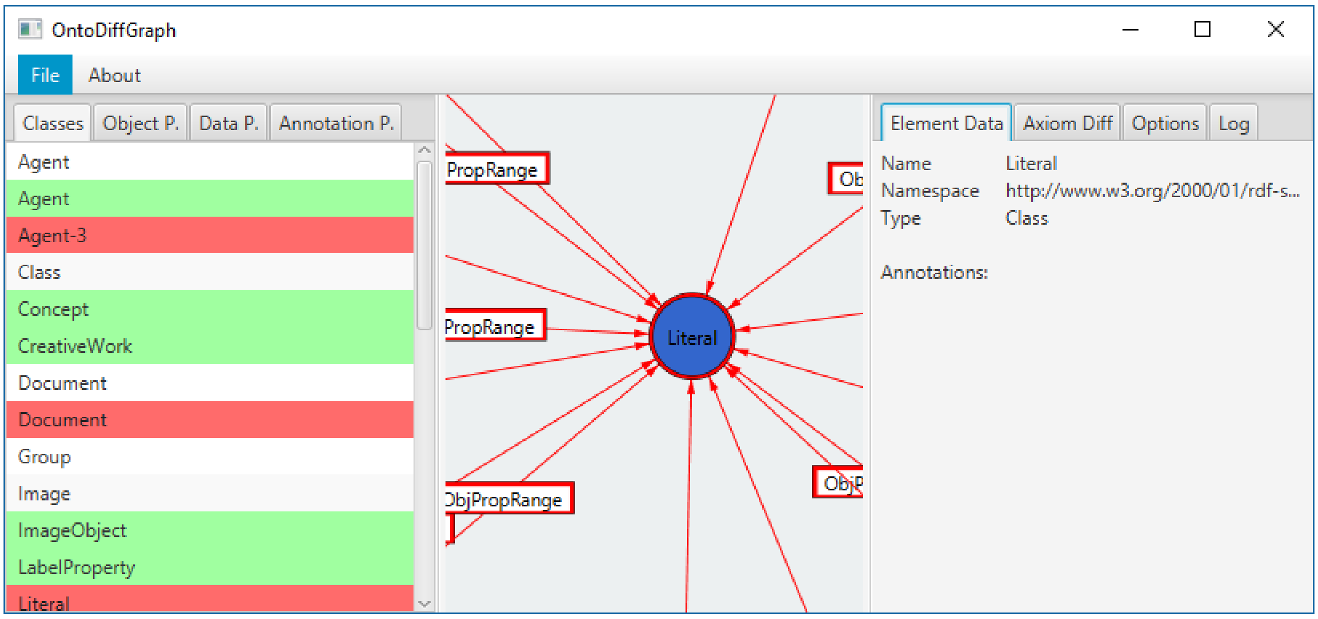
Displaying the difference

To display the difference between two ontologies the user needs to:

- Create the ontology diff (File > Create Ontology Diff)
- Select the old ontology version
- Select the most recent ontology version

Detecting differences

This application is divided in three separate interface elements (Left pane, Canvas, Right pane) as seen in Figure 49.



Name Literal
Namespace http://www.w3.org/2000/01/rdf-s...
Type Class

Annotations:

Figure 49.: OntoDiffGraph User Interface

In the left pane it is possible to browse the list of elements of the ontology: Classes, Object Properties, Data Properties and Annotation Properties. Each element in these lists has a background colour that represents what happened to that element from one version of an ontology to the other.
The colours have the following meaning:

- Red, this element was removed from the ontology
- Green, this element was added to the ontology
- White, this element was not changed

The middle section of the interface is where the graph containing all the elements of the ontology will be displayed. In order to understand the graph elements the user needs to read the graph notation contained in the appendix. It is possible for the user to interact with the graph by:

- Using the mouse wheel to zoom in and out
- Left clicking a graph node to display its information in the right pane
- Left clicking a graph node and dragging the mouse to move the node
- Right clicking and dragging to move the entire graph

In the right pane it is possible to browse details about an element, the list of axiom differences, the graph layout options, and the application log.

To display information in the tab "Element Data" the user needs to click on an element of the graph, however not all elements show information. The information shown depends on the element clicked, classes and properties will display their name, namespace, element type and annotations to the user. Anonymous elements will display their axiom representation to the user.

The "Axiom Diff" tab shows the user all the axioms of the ontology, all the axioms also use the same colour pattern as the elements in the left pane. In this tab it is also possible to filter the axioms by typing in the text field located at the top of the tab.

In the "Options" tab the user can change the default layout parameters. To view the difference in some ontologies the user might need to tweak some of these parameters. However the most important one and recommended is the "Show minimal view" option, which removes all elements in the graph that are not important if the user only wants to see what has changed.

The "Log" tab is used by the application to display information, warnings and errors to the user.

A.2 EXAMPLES AND QUESTIONS

EXAMPLES USING SIMPLE ONTOLOGIES

1. Classes and Properties (simple1-v1.owl and simple1-v2.owl)
   In this ontology it is possible to observe the following changes:
   - Removal of class A1, object property B1 and data property C1
   - Creation of class A3, object property B3 and data property C3
2. Domains and Ranges (simple2-v1.owl and simple2-v2.owl)
   In this ontology it is possible to observe the following changes:
   - Object property B1 - Class A1 removed from domain, class A3 removed from range, class A1 added to range
   - Object property B2 - Class A1 and class A2 added to domain
   - Data property C1 - Class A1 added to domain
   - Data property C2 - *int* removed from range and no longer used in the ontology
3. Other Elements (simple3-v1.owl and simple3-v2.owl)
   In this ontology it is possible to observe the following changes:
   - Object properties B5 and B6 were added to the ontology
   - Object property B1 - is no longer a reflexive property, is now a transitive property, is now disjoint with object property B5
   - Object property B3 - is now an inverse property of object property B4, is now equivalent to object property B6
   - Object property B5 - is now a subproperty of object property B4

EXERCISES

In these exercises we will be using the ontology FOAF (foaf-v1-20071002.rdf and foaf-v2-20091215.rdf). Please answer the following questions using the tool referenced in the question text.

1. [Protégé and OWLDiff] What is the object property that was added to the ontology?

   ____

2. [Protégé and OWLDiff] How many data properties were added that have the range *Literal*?

   ____

3. [Protégé and OWLDiff] What is the name of the external class (imported from another ontology) that was added to the FOAF ontology?

   ____

4. [OntoDiffGraph] What is the object property that was removed from the ontology?

   ____

5. [OntoDiffGraph] What is the name of the functional data property that was added to the ontology?

   ____

6. [OntoDiffGraph] What are the names of the external classes (imported from another ontology) removed from the FOAF ontology?

   ____

In these exercises we will be using the ontology MOD (mod-v1-31072015.owl and mod-v2-09092016.owl). Please answer the following questions using the tool referenced in the question text.

1. [Protégé and OWLDiff] What happened to the domain of the data property *creationDate* ?

   ____

2. [Protégé and OWLDiff] How many *subClassOf* axioms were added from the class *Ontology*?

   ____

3. [Protégé and OWLDiff] The class *Person* was added to the range of which property?

4. [OntoDiffGraph] What happened to the domain of the object property *evaluatedBy*?

5. [OntoDiffGraph] How many *subClassOf* axioms were removed from the class *Ontology*?

6. [OntoDiffGraph] The class *Project* was added to the domain of which data property?

FINAL QUESTIONS

1. How would you rate the following tools in a scale from 1 - 10?

OWLDiff	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 ☐
OntoDiffGraph	1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 ☐

2. How would you rate the your proficiency using Protégé in a scale from 1 - 10?

   1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 ☐

3. Have you worked with the OWLDiff plugin before taking this comparative evaluation?

   Yes ☐ No ☐

4. Cross the checkbox that represents how do you feel about the following statements:

   The graph helps the user visualize the difference between versions of the ontology.
   ☐ Strongly Agree ☐ Agree ☐ Neutral ☐ Disagree ☐ Strongly Disagree

   The graph notation is easy to understand.
   ☐ Strongly Agree ☐ Agree ☐ Neutral ☐ Disagree ☐ Strongly Disagree

   OWLDiff users also need to rely on Protégé to be able to understand some of the information displayed in the plugin interface.
   ☐ Strongly Agree ☐ Agree ☐ Neutral ☐ Disagree ☐ Strongly Disagree

5. Are there features that you think OntoDiffGraph is lacking? In this case, which ones?

   ____
   ____
   ____
   ____