Visualization of Ontology Evolution Using OntoDiffGraph*

André Lara[1], Pedro Rangel Henriques[2], and Alda Lopes Gançarski[3]

1 **Centro ALGORITMI, Universidade do Minho, Braga, Portugal**
`a64362@alunos.uminho.pt`
2 **Centro ALGORITMI, Universidade do Minho, Braga, Portugal**
`prh@di.uminho.pt`
3 **Institut Telecom, Telecom SudParis, CNRS Samovar, Evry, France**
`alda.gancarski@telecom-sudparis.eu`

Abstract

Ontologies evolve with the passing of time due to improvements, corrections or changes in requirements that need to be made. In this paper we describe a thesis work aiming at the creation of a visualization technique with the objective of allowing the viewer to easily identify changes made in an ontology. With the use of a specification based on the already existing Visual Notation for OWL Ontologies (VOWL) it is possible to display the differences that exist between two versions of an ontology. The proposed approach will be implemented in an application, that is also discussed in the paper.

1998 ACM Subject Classification H.3.1 Content Analysis and Indexing

Keywords and phrases Ontology Evolution, Ontology Visualization

Digital Object Identifier 10.4230/OASIcs.SLATE.2017.14

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 change 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 [3]. In order to map ontology elements to graphical entities there are already existing notations, one example of this is the Visual Notation for OWL Ontologies (VOWL) notation [9].

The Friend of a Friend (FOAF) Ontology[1], is an ontology that contains information about 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 [7, 10, 5] 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

* This work has been supported by COMPETE: POCI-01-0145-FEDER-007043 and FCT – Fundação para a Ciência e Tecnologia within the Project Scope: UID/CEC/ 00319/2013.
[1] Available from `http://xmlns.com/foaf/spec/`.

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.
The project here discussed 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. They can easily select the desired changes to visualize them directly in the ontology display window and a textual description.
- Implementation of this visualization technique in an application using Java.

This thesis research hypothesis is that through the use of the proposed technique, a visual *diff* using a graph to display the ontology and the changes made to 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.
This paper is divided in the following sections:

- Introduction: In this section we introduce the reader to the context of this paper and what it aims to achieve.
- Related Work: This section identifies and discusses work related to the proposal here reported. A broad and deep research was made on the topics concerned with this project: visualization of ontology evolution, and change detection. However for the sake of space it is not possible to include here all the material discovered; to read all the information collected and organized, the reader should see [8].
- OntoDiffGraph: The Proposal: In this section we discuss the proposed technique and the architecture of the application where it is going to be implemented, OntoDiffGraph.
- OntoDiffGraph: Development: This section contains information related to features and decisions that were taken during the development of the application.
- Conclusion: In this section, we analyse what has been described in this paper and the future work that still needs to be done.

2 Related Work

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 [1]), built on the already existing Resource Description Framework (RDF [2]) and RDF Schema (RDFS [4]) specifications.

Visual Notation for OWL Ontologies (VOWL) [9] 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 1 it is possible to observe the representation of the various elements of an ontology. This notation can be read in more detail in [9].

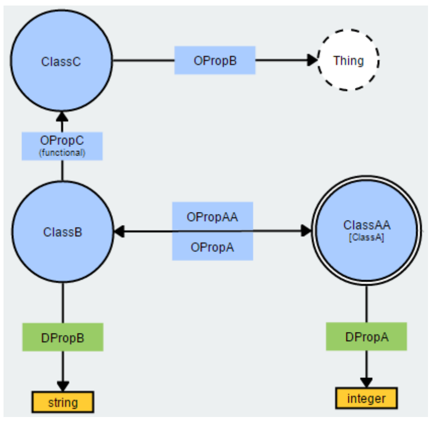

Figure 1 Visualization of an ontology with the VOWL notation.

2.2 Change Visualization

PROMPT-Viz [11] 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.

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 [12] 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

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 [9], used to map ontology elements to visual graph elements in order to easily display an ontology, with the objective of displaying the differences between two versions of an ontology.

To make the user able to easily identify what has been changed and the type of change made to an element we decided to add borders to the visual representation of the various ontology elements. The types of changes that will be identified are creations, modifications and deletions. Each one of these types will have a colour associated with them: creation will

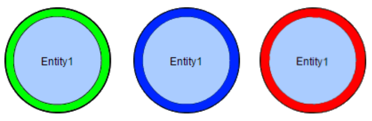

Figure 2 Class Creation, Modification and Deletion.

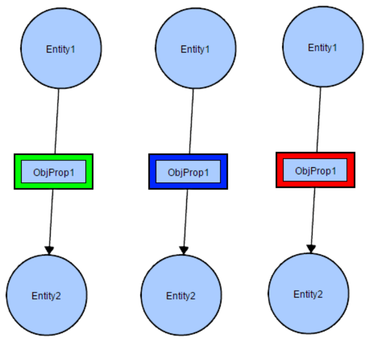

Figure 3 Relation Creation, Modification and Deletion.

have a green border, modification will have a blue border and deletion will have a red border. Figure 2 shows what a owl:Class node would look like when displayed with this notation.

Figure 3 shows what the edges would look like when the notation is applied on a owl:ObjectProperty.

Changes to the domain or range of the object property will make this element have a blue border, because these changes are considered a modification of the property. The change in domain and range will also be seen in colours of the edges connecting the property to its targets. Figure 4 displays a change in the object property domain and range, the domain and the range used to be **Resource** but in the new version it is **Thing**.

In Figure 5 it is possible to observe what the edges would look like when the notation is applied on a owl:DatatypeProperty.

However, due to the large size of some ontologies, it can still be difficult to locate the changes existing from one version to the other one, even with the added visual information. To solve this, the user will also have access to the textual version of the changes made in the ontology. When the user selects one of the changes in the textual version the viewable area in the application will reposition itself so that the change in centered on that area.

OntoDiffGraph: Architecture

The previously stated features are being implemented in a Java program, using JavaFX to draw the graph and OWL API [6] to extract information from the ontology.

Figure 6 contains the architecture of the proposed system. This system will be able to load ontologies and display the differences between them through the use of a graph.

To load ontologies, we use of the OWL API [6], a Java API used to create, parse, manipulate and serialize OWL ontologies. This process can be observed in Figure 6 process named *Load Ontologies*. After the loading task the system compares the ontologies that were loaded and identifies the differences that exist between them, in the *Calculate Differences* process. With this information, the system will be able to generate a graph in the *Generate Graph* process. This graph contains the ontology information and the differences between the ontologies. After creating the graph structure, the only step left to do is drawing the

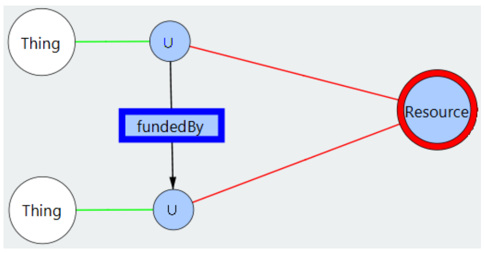

Figure 4 Example of a change in domain and range.

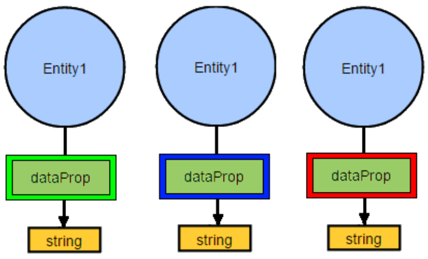

Figure 5 Datatype Creation, Modification and Deletion.

graph so that the results can be displayed to the user. This will be done with the use of a JavaFX Canvas in the *Draw Graph* process.

4 OntoDiffGraph: Development

This section contains detailed information about the features and decisions made in the development of the application. It does not only contain the implementation of the technique but all the necessary features for a complete ontology visualization application, such as manipulation of graph elements, layout customization or the visualization of the ontology.

4.1 Basic features

Some important features need to be added to improve the usability of the application. The following features 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 move the visible graph area of the application.
- Dragging, allows the users to drag nodes of the graph to their desired position.

It will also be 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 and data properties of the ontology. By selecting an element of the list, the node in the visualization will be centered in the viewable area. This menu is shown in the following figure:

In this menu it is also possible to see the elements of the ontology that were added, modified and 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.
- Blue, the element was modified.
- Red, the element was removed from the ontology.

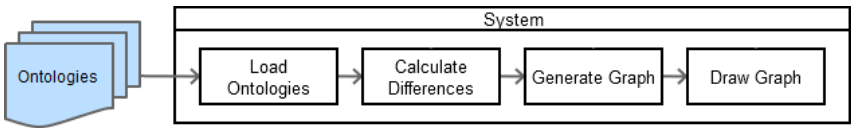

Figure 6 Architecture of the OntoDiffGraph system.

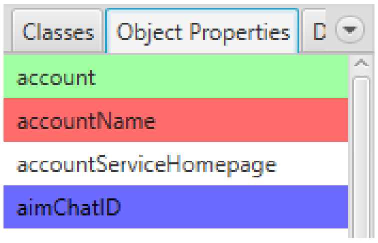

Figure 7 Menu with all ontology elements.

4.2 Layout 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 to these.

The currently available variables that can be changed in the force-directed layout are the following:

- 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.
- Damping, controls the damping of the forces of the graph.
- Timestep, controls the seconds that pass in every iteration of the algorithm.

4.3 Change detection

In order to find all the differences between two versions of an ontology it is necessary to compare its contents. All ontology elements contain an unique IRI that identifies them, this will help identify elements that were added or removed. The following list contains the change 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.
- Element was modified, an element is classified as modified if information contained in this element is modified. An example of this is a change in the element domain or range as seen in Figure 4.

However this method has some weaknesses, one of those is that it does not classify a change in an element name as a modification. The renaming of 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.

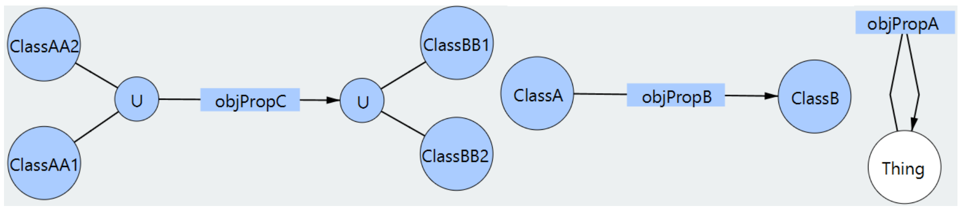

Figure 8 Visualization of classes and object properties in the application.

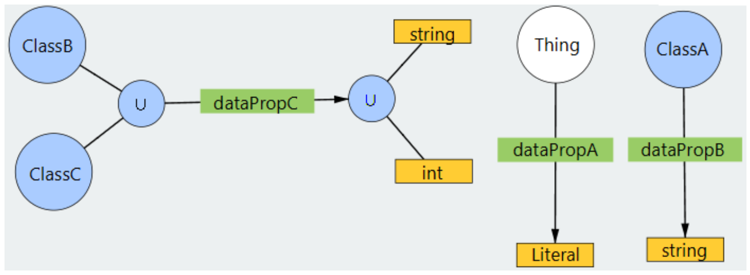

Figure 9 Visualization of classes and data properties in the application.

4.4 Visualization

In order to create the visualization it is necessary to obtain all the elements of the ontology and transform these into nodes and edges with their respective visual representation. The following figures displays classes, object and data properties as seen in the developed application.

In the graph it is also possible to observe the union node that is created when the property domain or range contains more than one element.

5 Conclusion

In this paper the context, motivation and objectives of the underlining work were presented and discussed. The outcomes concerning the research done on the topics involved in the state of the art of our working area were reported. Concerning the topic of ontology visualization we justified the adoption of some of the visual elements from VOWL with the fact that it is a good notation to display ontologies and that it is already being used by several applications. Tools that allow the visualization of the difference between ontologies and their main strengths and weaknesses were also discussed – such survey supported our decisions in the creation of OntoDiffGraph. To solve this problem we propose a solution that consists of a visualization technique that will be implemented in an Java application. The architecture, features and development decisions of this application were detailed in this paper.

The application that was proposed is still under development, however the visualization of an ontology is almost complete. There is still work that needs to be done in the detection of differences in two versions of an ontology and the representation of this information however it is already possible to determine the ontology elements that were added and removed from the different versions. After the implementation of these features in the application it will be evaluated to verify if it needs to be modified or improved.

References

1 Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. *OWL Web Ontology Language Reference*. World Wide Web Consortium, 2004. `http://www.w3.org/TR/2004/REC-owl-ref-20040210/`.

2 Dave Beckett. *RDF/XML Syntax Specification (Revised)*. World Wide Web Consortium, 2004. URL: `http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/`.

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

4 Ramanathan Guha and Dan Brickley. *RDF Schema 1.1 Recommendation*. World Wide Web Consortium, 2014. URL: `http://www.w3.org/TR/2014/REC-rdf-schema-20140225/`.

5 Michael Hartung, Anika Gross, and Erhard Rahm. CODEX: exploration of semantic changes between ontology versions. *Bioinformatics*, 28(6):895–896, 2012.

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

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

8 André Lara. Visualization of Ontology Evolution using OntoDiffGraph. Master's thesis, University of Minho, 2017.

9 Steffen Lohmann, Stefan Negru, Florian Haag, and Thomas Ertl. VOWL 2: user-oriented visualization of ontologies. In Krzysztof Janowicz, Stefan Schlobach, Patrick Lambrix, and Eero Hyvönen, editors, *International Conference on Knowledge Engineering and Knowledge Management*, pages 266–281, 2014.

10 Natalya Fridman Noy, Mark A. Musen, et al. PromptDiff: A fixed-point algorithm for comparing ontology versions. In *Association for the Advancement of Artificial Intelligence (AAAI) Conference*, pages 744–750, 2002.

11 David Stephen John Perrin. *Prompt-viz: Ontology version comparison visualizations with treemaps*. PhD thesis, University of Victoria, Canada, 2004.

12 Miguel Ángel Rodrıguez-Garcıa, Luke Slater, Keiron O'Shea, Paul N. Schofield, Georgios V. Gkoutos, and Robert Hoehndorf. Visualizing ontologies with AberOWL. In James Malone, Robert Stevens, Kerstin Forsberg, and Andrea Splendiani, editors, *Semantic Web Applications and Tools for the Life Sciences*, pages 183–192, December 2015.