GDAŃSK UNIVERSITY OF TECHNOLOGY FACULTY OF **ETI ANNALS**
No 8 Information technologies 2010

Tomasz Boiński, Anna Jaworska, Radosław Kleczkowski, Piotr Kunowski

Gdańsk University of Technology, Faculty of Electronics, Telecommunication and Informatics, Departament of Computer Architecture

ONTOLOGY VISUALIZATION*

Abstract

Ontologies are often represented as a graph where nodes represent objects within given ontology and edges represent relations between those objects. Such graphs can be very complex even for medium sized ontologies so a clean and complete graphical representation is needed. Existing solutions do not include all elements or produce unreadable results for big ontologies. In this paper a complete set of graphical objects for ontology representation was proposed. Algorithms for graph creation were defined. An implementation of proposed solution is described. Comparison with other solutions is made.

1. INTRODUCTION

Creation of proper ontologies is a complex task. The amount of triples even for small solutions can be big. To be able to extend ontologies easily a mean for its visualization, which enhances ability to perceive and estimate created ontology, is needed. There are existing solutions in this field, mainly representing ontologies as a Explorer-like tree or a graph [1]. Tree-like structures have problems when a class have many parents – it appears multiple times in the hierarchy introducing possibility of a mistake when a human is interpreting such visualization.

Graph based visualizations can present ontologies in clearer way. Solutions like Jambalaya [2] or OWLViz [3], created as a plug-in for well known and popular ontology editor – Protégé [4][5][6], have many visualization methods and filters implemented. Readability of visualization of different properties via edges with labels is poor due to overlapping of those labels. Classes are always represented by big squares which too limits readability and requires a lot of display space. The visualization in Jambalaya is based on a hierarchy relative to chosen relation. When this hierarchy is degenerated (very wide or very deep) visualization of whole ontology becomes very big and user cannot take a peek on all relations in the ontology at once. Furthermore currently this plug-in cannot be used with newest 4th version of the Protégé editor.

* This work was supported by tasks performed within N519 432338 MNiSW grant

Other well known solution is GrOWL [7]. A rather simple solution that became a base for authors of this paper when developing presented visualization concept. GrOWL don't have limitations known from Jambalaya. Unfortunately for operations on ontologies utilizes Jena [8] framework, which introduces need of using heavy library dependencies.

In this paper a solution for ontology visualization in Ontology Creation System (OCS) [9] is described. Similarly to OCS the proposed solution is designed for ontologies stored using OWL DL dialect of OWL language [10] and processed using OWL API [11] library thus eliminating usage of heavy frameworks.

In the following section a set of symbols used for visualization of ontology elements is proposed. In section 3 visualization algorithm is described.

2. VISUALIZATION SYMBOLS

OWL language specification allows wide usage of its concepts for defining ontology elements and its restrictions. It is important for visualization software to produce results in pair to used constructs keeping this way synchronization between ontology and its visualization. Produced image needs to be clear and easy to comprehend and interpret as well. Due to large number of possible elements it is important that used graphical notation allows clear and fast distinction between those elements.

To fulfill above mentioned conditions a set of symbols representing all possible concepts of OWL DL language was proposed.

Symbols representing Classes, Properties and DataTypes all have a shape of a rectangle with round edges as they all share same types of relations. For further distinction they are different in terms of used color and angle of the edge rounding. An Individual is represented by a rectangle as it takes part in different set of relations (Fig. 1).

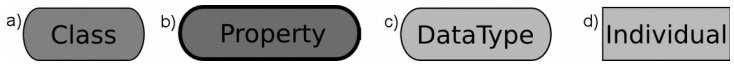

Fig. 1. Symbols representing a Class (a), Property (b), DataType (c) and an Individual (d)

The biggest challenge in ontology visualization is representation of anonymous classes. In this solutions complex relations introducing anonymous classes are presented with a circle containing meaningful character inside. Sample symbols are shown in Fig. 2. In case of intersection, complementarity, union and cardinality mathematical symbols were used: ∩, ¬, ∪ and N respectively. Same rule was applied for relations between Individuals: = symbol represents "sameAs" relation and ≠ symbol represents „allDifferent" and "differentFrom" relations. Symbol for „sameAs" relation is redundant but kept for cohesion of visualization as this relation if reverse to "differentFrom" and "allDifferent" relations. In case of cardinality relation another node is added that represents min, max and equal cardinality.

Relations „allValuesFrom" and „someValuesFrom" are displayed by introduction of an anonymous class representing the result of given restriction (Fig. 3). This class is connected with given property. The name of the property is preceded by general quantifier symbol for „allValuesFrom" restriction and existential quantifier symbol for „someValuesFrom". An arrow leading from the property points to a class, which can be any simple named class or other complex anonymous class, or an Individual that is a counter-domain for given

property. Additionally the property is connected with a node representing definition of given property.

Simple relations are represented by lines with different shafts. In general direction of the arrowhead points to a direction in which given axiom should be read. Where it is not necessary lines do not have shafts to increase readability of the whole picture. Some of the relations were presented in Fig. 4.

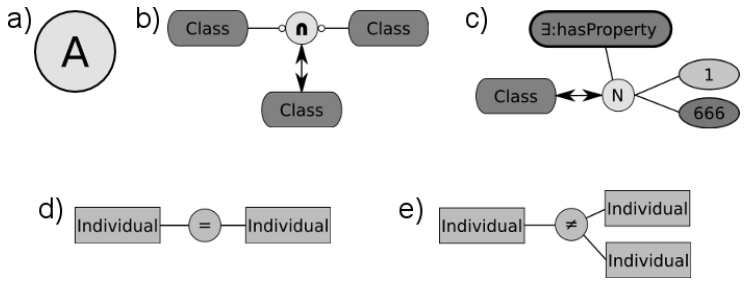

Fig. 2. Example of symbols representing anonymous class (a), intersection (b), min and max cardinality (c), sameAss relation (d) and allDifferent relation (e)

Relations “subClass” and “subProperty” are represented in the same manner as in UML language – as an arrow with an empty shaft. Edges that represent “equivalent” and “disjoint” relations have shafts pointed in opposite directions stressing the reversibility of that relations. In case of Property definition inversity is marked by red color of the arrow and marked in two different way depending on the fact whether given property is symmetric or asymmetric. Equality of properties is distinguished form equality of the classes by a different color.

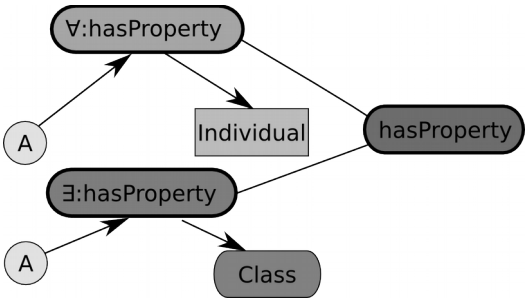

Fig. 3. someValuesFrom and allValuesFrom relations representation

To be able to distinguish “range” and “domain” relations attached to a property two additional shaft were introduced. They are presented in Fig. 4e and 4f.

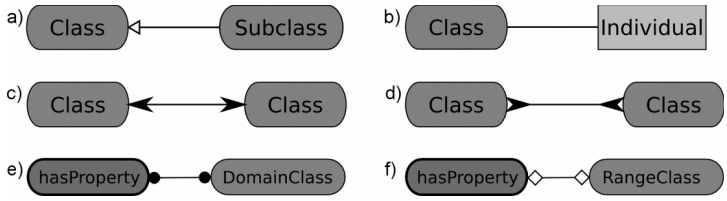

Fig. 4. Symbols representing simple relations: rdfs:subclassOf (a), instanceOf (b), owl:equivalentClass (c), owl:disjointWith (d), rdfs:domain (e) and rdfs:range (f)

3. VISUALIZATION ALGORITHM

The amount of possible elements and versatility of their usage implies high complexity of the algorithm used for ontology visualization. In this paper an algorithm able to visualize ontology of any complexity, keeping its structure and used elements, is proposed.

```
(1)begin
(2)     insert basic elements;
(3)     for each axiom begin
(4)             if property axiom begin
(5)                     insert anonymous node;
(6)                     insert connections between nodes;
(7)             end if
(8)             if individual axiom begin
(9)                     insert anonymous node;
(10)                    insert connections between nodes;
(11)            end if
(12)            if class axiom begin
(13)                    if description axiom begin
(14)                            start procedure InsertDescription;
(15)                    end if
(16)                    insert connections between nodes;
(17)            end if
(18)    end for
(19)    insert owl:Thing element;
(20)    connect not superclassed classes with owl:Thing;

(21)    Procedure InsertDescription
(22)    begin
(23)            if someValuesFrom or allValuesFrom
                        or hasValue or  cardinality axiom begin
(24)                    insert property usage node;
(25)                    insert edges;
(26)                    start procedure InsertDescription;
(27)            end if
(28)            if setTypeAxiom begin
(29)                    for each descritpion node begin
(30)                            start procedure InsertDescription;
(31)                    end foreach
(32)            if class or individual begin
(33)                    insert connection;
(34)            end if
(35)    end
(36)end
```

Alg. 1: Algorithm generating the graph visualizing the ontology

In general the algorithm visualizes first all basic elements, like classes, properties and individuals, as nodes. Then runs through all axioms and relations connected with them introducing proper anonymous classes and edges representing those relations. Finally usage of some/allValuesFrom and hasValue is introduced into the visualized graph. Detailed flow of the algorithm is presented in the Alg. 1.

After introduction of the main nodes(2) list of all axioms is looked through(3). Some cases can be distinguished. For axioms defining attributes of a property(4): functional, inverseFunctional, symmetric and transitive, proper anonymous classes(5) and edges connecting them with appropriate property(6) are inserted into the graph. Relations like inverseProperty, equivalent properties and subProperty are also inserted into the graph according to symbols used for the visualization. In the same way axioms defining relations between individuals (different, allDifferent and same individuals) are treated(8-11).

In case of the axioms related to classes(12) the algorithm needs to take into consideration that the class can be defined by a complex description(13-15). Such cases are axioms defining relations between classes: equivalentClasses, subClass, disjointClasses etc. Moreover such dependencies can occur in case of a class definition based on range and domain attribute of a property and in cases like classAssertion, where a class being the type of given Individual is defined. When such relations consider only directly defined class, appropriate edge is inserted into the final graph connecting given class with other node(16). Aforementioned relations are created using proper anonymous classes which in turn are represented by predefined symbols.

When an axiom represents hasValue, someValuesFrom or allValuesFrom(23) relation (where hasValue is the same as allValuesFrom, but with an Individual, not class, as a target node) firstly an anonymous class is inserted into a graph. If this class represents cardinality restriction its symbol is changed appropriately (into N)(28) and nodes with numbers representing min, max or equal cardinality are added. Next proper edge and a node representing usage of given property, edge connecting that usage with definition of a property and an edge with an arrow pointing to a value defined by the axiom. This value can be either an Individual, a class or a description so a recursive execution of the same actions is needed. When an axiom defines set of classes or set of descriptions like unionOf, intersectionOf, complementOf(28) for each element of the set(29) an edge needs to be added into the graph and above mention procedure executed for target of this relation.

Another example is the case when a class is defined by oneOf axiom. In this case proper anonymous class is inserted into a graph and connected with related Individuals.

Above mentioned principles are also applied to DataTypes. In this case all rules regarding classes are related to given DataType.

Most of the ontologies do not define subClass relations with owl:Thing class. In this case to keep consistency of produced graph each class that is not explicitly connected with other class with a subClass relation is connected with owl:Thing(19-20) as its superclass. Its the only deviation from the rule of explicitly visualizing that what was defined by the author of the ontology.

4. CONCLUSIONS

Proposed solution allows visualization of ontologies performed as a graph. Ontologies of any complexity can be visualized. Thanks to usage of Prefuse [12] library, graphical layout of graph elements is automatically optimized in the means of its gravity mechanism. User can at any time turn off this behavior and move any of the elements by hand to

increase readability. Size of the elements is dependent on length of its label and visual zoom which combined with different visual filters makes possible the visualization of the most important elements only. Preliminary tests performed by students and 3 experts from staff of GUT's Department of Computer Architecture show good readability for both normal and color blind recipients. Further tests will allow fine tuning of proposed symbols.

BIBLIOGRAPHY

[1] Katifori A., Halatsis C., Lepouras G., Vassilakis C., Giannopoulou E., *Ontology Visualization Methods – A Survey*, ACM Computing Surveys, Vol. 39, No. 4, Article 10, 2007
[2] Storey M., Lintern R., Ernst N., Perrin D., *Visualization and Protégé*, http://protege.stanford.edu/conference/2004/abstracts/Storey.pdf, 2004
[3] Horridge M., *OWLViz*, http://code.google.com/p/co-ode-owl-plugins/wiki/OWLViz, 2010
[4] Stanford Center for Biomedical Informatics Research, http://protege.stanford.edu/, 2009
[5] Gennari J. H., Musen M. A., Fergerson R. W., Grosso W. E., Crubézy M., Eriksson H., Noy N. F., Tu S. W.: *The evolution of Protégé: An environment for knowledge-based systems development*, Technical Report SMI-2002-0943, Stanford Medical Institute, 2002
[6] Noy N. F., Fergerson R. W., Musen M. A.: *The knowledge model of Protégé-2000: Combining interoperability and flexibility*, Lecture Notes in Computer Science, Springer-Verlag, 2000
[7] Krivov S., Williams R., Villa F., *GrOWL: A tool for visualization and editing of OWL ontologies*, Web Semantics: Science, Services and Agents on the World Wide Web , Vol 5, Issue 2, 2007, p. 54-57, 2007
[8] Carroll J. J., Reynolds D., Dickinson I., Seaborne A., Dollin C., Wilkinson K., *Jena: Implementing the Semantic Web Recommendations*, Digital Media Systems Laboratory, HP Laboratories Bristol, HPL-2003-146, 2003
[9] Boiński T., Budnik Ł., Jakowski A., Mroziński J., Mazurkiewicz K., *OCS - domain oriented Ontology Creation System*, Sejmik Młodych Informatyków, Polish Journal of Environmental Studies, Vol 18, No. 3B, 2009, p. 35-38, 2009
[10]W3C, *Web Ontology Language (OWL)*, http://www.w3.org/2004/OWL/, 2004
[11]Horridge M., Bechhofer S., *The OWL API: A Java API for Working with OWL 2 Ontologies*, 6th OWL Experienced and Directions Workshop, 2009
[12]Heer J., Card S. K., Landay J. A., *Prefuse: a toolkit for interactive information visualization*, Proceedings of the SIGCHI conference on Human factors in computing systems, p. 421-430, Portland, Oregon, USA: ACM, 2005

WIZUALIZACJA ONTOLOGII

Streszczenie

Ontologie bardzo często są reprezentowane w postaci grafu, którego wierzchołkami są obiekty występujące w prezentowanej ontologii a krawędziami relacje i powiązania pomiędzy tymi obiektami. Grafy reprezentujące ontologie mogą być złożone już dla średniego rozmiaru ontologii. Stąd niezbędna jest pełna i czytelna reprezentacja elementów takiego grafu. Dostępne rozwiązania zazwyczaj nie oferują reprezentacji pełnego zbioru elementów lub są nieczytelne w przypadku dużych ontologii. W publikacji zaproponowano zbiór symboli graficznych reprezentujących wszystkie elementy ontologii. Przedstawiono algorytmy konstrukcji grafu na podstawie obiektowej reprezentacji ontologii. Opisano implementację zaproponowanego rozwiązania oraz porównano ją z innymi dostępnymi rozwiązaniami.