

# A Visual Language for OWL Lite Editing

Jonas Rebstadt, Luisa Brinkschulte, Arkadij Enders, Robert Mertens

Dept. of Computer Science  
HSW University of Applied Sciences  
Hameln, Germany

{jonas.rebstadt|luisa.brinkschulte|arkadij.enders|mertens}@hsw-hameln.de

## ABSTRACT

While ontologies are a powerful means for knowledge representation, they are not as wide spread as one would think. A major reason for this fact is the lack of an intuitive visual editor. The visual language presented in this poster tackles part of this problem in that it was developed for a visual editor that allows on-line consistency checking of visual editing steps, i.e. preventing to describe instance configurations that are not allowed by the corresponding class descriptions. The visual language is closely linked to the editor's data model and covers all elements of OWL lite. It is comprised of twelve elements in total, two node elements and ten link types. The language has been implemented in a prototype version of the editor and used to visualize a number of publicly available OWL lite ontologies.

## CCS Concepts

• **Information systems~Web Ontology Language (OWL)**  
• **Information systems~Ontologies** • *Information systems~Data structures*  
• **Human-centered computing~Visualization**  
• **Computing methodologies~Ontology engineering**  
• *Computing methodologies~Knowledge representation and reasoning*  
• *Software and its engineering~Visual languages*

## Keywords

Ontology; Visualization; OWL Lite; Knowledge Representation

## 1. INTRODUCTION

Ontologies are used as a human-understandable and editable means for knowledge representation. The advantages of ontologies range from re-use and exchange of knowledge between people to automatic reasoning and search support.

More and more users are interested in ontologies because of the increasing importance of electronic data exchange between institutions. Not only experts of knowledge representation but also a high number of non-experts dealing with ontologies use them as a means to exchange, edit and structure knowledge.

Aside from the typical fields of artificial intelligence (AI), databases and web-technologies, there are also numerous other fields from science and industry which get in contact with ontologies. To support groups which are less familiar with ontologies, visualization plays an important role.

Current editors for the visualization of ontologies are characterized by their comprehensive presentation of all functions and therefore hard to read for non-experts. To make ontologies useable and editable for non-experts it is even more important to de-

scribe the complex functions on a higher level of abstraction.

The solution to this is an intuitive and interactive visualization tool which abstracts the variety of language elements in the background of OWL from the actual application. With these goals we explored the possibility to use mind map like visualization as base for a visual editor that allows users to create and work with ontologies in a simplified and intuitive way [1]. One necessary step for such a tool is a well-specified visual language for user-oriented representation of ontologies.

## 2. Related Work

Protégé [2] is the best-known and most comprehensive ontology editor and was developed at Stanford University. Other widely used ontology editors are SWOOP by Mindswap, OntoStudio by ontoprise GmbH, Apollo by Knowledge Media Institute and TopBraid Composer by TopQuadrant [2, 3].

OWLGrED is a visual authoring tool for OWL using a mix of UML and textual syntax [4]. It allows visualizing ontology fragments in order to edit large ontologies. The tool does not provide edit-time consistency checking however it comes with zooming functionality. The OWLViz<sup>1</sup> plugin for Protégé [5] contributes ontology visualization but neither graphical editing nor visualization of properties [6]. A complete consistency check only happens after explicitly running a reasoning engine. Zooming is possible, and some elements (classes but no connections) can be hidden in the visualization. OntoGraf<sup>2</sup> is a standard graph visualization plugin of Protégé [7]. It is limited to visualize ontologies, hence it does not allow to edit them. Therefore, it does not come with edit time consistency checking. It does however, offers zoom capabilities. OntoGraf is capable of filtering notes and connections.

The use of intuitive notations and easily understandable notation symbols, colors and node shapes are covered by GrOWL [8] and the Protégé plugins SOVA<sup>3</sup> and VOWL<sup>4</sup> [9, 10]. GrOWL and SOVA are rather designed for users with expertise in description logic and related symbols. VOWL provides graph visualizations to represent ontologies for users less familiar with ontologies and is therefore most closely related to our developed visual language. Contrary to VOWL, the developed visual language focuses on combining class- and individual views. Furthermore it provides user-friendly support by intuitive operations and element connections. While VOWL uses the same visual element for all properties and one can only distinguish the type of the property by read-

<sup>1</sup> <http://protegewiki.stanford.edu/wiki/OWLViz>

<sup>2</sup> <http://protegewiki.stanford.edu/wiki/OntoGraf>

<sup>3</sup> <http://protegewiki.stanford.edu/wiki/SOVA>

<sup>4</sup> <http://protegewiki.stanford.edu/wiki/VOWL>







ing the label [10], we developed visual elements for each property. In addition to the objective of making ontologies understandable, we are actively working towards making ontologies editable in the course of the visual representation.

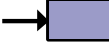


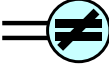
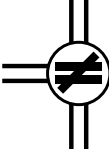

### 3. Visual Representation-Language for OWL Lite Elements

#### 3.1 Language Elements

Creating a new ontology isn't easy, especially if the raw owl language is used, but yields a high formalization and semantic expressiveness which can be used by the computer. However, mind maps are common, easy to use and intuitive. Hence, we tried to combine the strengths of both worlds to create ontologies easier and keep the owl data structure. The following table shows the view elements, which are used in our prototype editor to build an ontology with their use in the OWL light language.

Table 1. Language Elements

Name	View element	Description
Class		A round rectangle represents a defined owl:class of the ontology. The name of the class can't exceed four lines on the visual screen, but three dots at the end indicate that the actual name is longer, even though a class name should be short and precise. Furthermore the background and text color can be changed by the user.
Individual		The owl:individual is also represented as a round rectangle. Since the shape is exactly like the shape of a class, a color code must be used to distinguish an individual from a class.
IsA		The ClassAssertion is represented by the isA connection. It can be used to connect classes by a directed connection or individuals with classes to determine their class.
AreSame		To determine the equality of two individuals or classes the AreSame connection is used.
HasValue		The owl:DatatypeProperty is represented by the HasValue connection. However, to set HasValues on an individual it is necessary to define them first at the class level. On creating a HasValue an owl or rdf datatype is chosen. Afterwards it is possible to add a value to an individual.
HasRelation		OWL:ObjectProperty is covered by the element HasRelation. By defining a relation between two classes, the individuals of these classes are able to have the defined relations as optional connections. After defining HasRelations, the user is able to choose which of the HasRelations exist between two individuals.

Name	View element	Description
Functional		An owl:FunctionalProperty is represented by an element of type Functional. Thus, users can define a Functional between two classes to connect certain individuals. An individual is only allowed to have one Functional per definition. '[...] a given individual Vintage can only be associated with a single year using the hasVintageYear property' <sup>5</sup>
Transitive		OWL:Transitive can be created by using the Transitive element. This element connects a class with itself to create a transitive relation. Afterwards two individuals of the same class can be connected by the Transitive tool.
Symmetric		The Symmetric tool is directed to both sides. Therefore the relation is bidirectional to indicate the relation works both directions. It represents owl:Symmetric in the ontology.
AreDifferent		AreDifferent covers the OWL property owl:DifferentFrom. It can be used between two individuals to determine that they are different from each other. It is possible to upgrade AreDifferent to an AllDifferent by using the AllDifferent tool and selecting an AreDifferent connection.
AllDifferent		To use owl:AllDifferent it is possible to create an AllDifferent object. This object looks almost like AreDifferent, however it is selectable. The user is allowed to add or remove connections between AllDifferent and individuals. Also it is possible to downgrade an AllDifferent to an AreDifferent by connecting only two individuals to the object.
Intersection		The key symbol shows if a class represents an intersection. Thus, the symbol can only be used between classes. By defining an intersection class the key is shown next to the isA edge and within the intersection class. The idea of the key symbol is that the connections can not be removed as the class is described by the incoming connections.

Not every owl light tag is mentioned in the above table, therefore table 2 shows every tag of OWL light and the visual elements. The X indicates which tag is used in which of the view elements.

Next to the visualized elements there are relevant owl/rdf(s)-elements that are only shown in the sidebar, as shown in the matrix below. An own visualization of these elements is in most cases not expedient as it would rather decrease the clarity and the intuitive handling. Based on these elements constructs like restrictions could be added in the sidebar, furthermore elements like this are influencing the creation and extension of ontologies but they are not visualized on their own.

<sup>5</sup> <https://www.w3.org/TR/owl-guide/>

Table 2: Mapping of OWL Lite tags to visual elements<sup>6,7</sup>

Visual Element OWL Lite Tag	Class	Individual	isA	AreSame	HasValue	HasRelation	Functional	Transitive	Symmetric	AreDifferent	AllDifferent	Intersection	SideBar
Rdf:Property			X	X		X	X	X	X	X			
Rdfs:Class	X												
Rdfs:domain	X	X	X	X		X	X	X	X				
Rdfs:isDefinedBy													X
Rdfs:label	X	X	X	X	X	X	X	X	X	X	X	X	X
Rdfs:range	X	X	X	X		X	X	X	X				
Rdfs:subClassOf			X										
Rdfs:subPropertyOf			X										
Owl:AllDifferent											X		
Owl:AllValuesFrom													X
Owl:cardinality													X
Owl:class	X												
Owl:DatatypeProperty					X								
Owl:DifferentFrom										X			
Owl:distinctMembers											X		
Owl:EquivalentClass				X									
Owl:EquivalentProperty				X									
Owl:FunctionalProperty							X						
Owl:intersectionOf												X	
Owl:inverseFunctionalProperty							X						
Owl:inverseOf													X
Owl:maxCardinality													X
Owl:minCardinality													X
Owl:nothing													
Owl:ObjectProperty						X							
Owl:onProperty			X	X	X	X	X	X	X				X
Owl:Ontology													X
Owl:OntologyProperty													X
Owl:Restriction													X
Owl:SameAs				X									
Owl:SomeValuesFrom													X
Owl:SymmetricProperty									X				
Owl:Thing													X
Owl:TransitiveProperty								X					

<sup>6</sup> The Tags Rdfs:comment, Rdfs:seeAlso, Owl:AnnotationProperty, Owl:backwardCompatibleWith, Owl:deprecatedClass, Owl:deprecatedProperty, Owl:imports, Owl:incompatibleWith, Owl:priorVersion and Owl:VersionInfo are visualized in the SideBar.

<sup>7</sup> An example of the visualization can be found at <https://www.researchgate.net/publication/306065998> A Visual Language for OWL Lite Editing

Next to the elements influencing the processing of ontologies other elements such as VersionOf provide additional relevant meta information.

### 3.2 Grammar

To allow a use as language for knowledge representation, the elements presented in the previous chapter are embedded in a grammar. The grammar is differed in the possible link types. It defines for every link type the possible elements that could be used before and after the considered link type. The grammar shown in the table below is limited to the visualized elements and presented according the EBNF. The realization of the other elements (for e.g. the restriction) is described in the following chapter.

Connection	Tupel (EBNF)
Intersection	Class, {Class}, 'Intersection', Class, {Class}
Are Same	(Class, 'AreSame', Class)   (Individual, 'AreSame', Individual)
Are different	(Class, 'AreDifferent', Class)   (Individual, 'AreDifferent', Individual)
All different	(Class, {Class}, 'AllDifferent', Class, {Class})   (Individual, {Individual}, 'AllDifferent', Individual, {Individual})
IsA	(Individual   Class), 'IsA', Class
Has value	(Class, 'HasValue', <String>)   (Individual, 'HasValue', <String>)
Has relation	(Class, 'HasRelation', Class)   (Individual, 'HasRelation', Individual)
Functional	(Class, 'Functional', Class)   (Individual, 'Functional', Individual)
Transitive	(Class, 'Transitive', Class (*The same one*))   (Individual, 'Transitive', Individual)
Symmetric	(Class, 'Symmetric', Class)   (Individual, 'Symmetric', Individual)

The above rules are identical to OWL and simply adapted to the visual language. For the Transitive connection on the class level, we have added the constraint that only elements of the same class can be connected as transitivity requires the same kind of relation. If users created two Transitive connections with the same name between three classes, the connections would still be two different connections, albeit of the same name.

### 3.3 Context based Constraints

As mentioned in chapter two there are elements influencing the creation and extension of ontologies not only in a direct but rather more in a context based way. Most important in this context is the correlation between classes and the respective individuals. The definitions and the link elements given for classes are not only

influencing the handling of this specific class. Rather they are determining how individuals of the defined class could be used and which link elements could be connected to them.

### 4. Conclusion and future work

The developed visual language allows users who are less familiar with knowledge representation and description logic to work with ontologies as a means to visualize, organize and edit knowledge in complex domains. It is successfully implementing all OWL Lite elements. However, the intuitive usage needs to be further validated in a user study in the future.

We're currently developing the already elaborated prototype in which a large part of the semantic, i.e. online consistency checking has already been implemented.

### 5. REFERENCES

- [1] L. Brinkschulte, A. Enders, J. Rebstadt and R. Mertens, "Aspect-Oriented Mind Mapping and Its Potential for Ontology Editing", *EEE Tenth International Conference on Semantic Computing (ICSC)*, pages 194 – 201, 2016.
- [2] E. Alatrish, „Comparison of Some Ontology Editors“ *Management Information Systems*, Vol. 8, Nr. 2, pp. 18-24, 2013.
- [3] V. Kashyap, C. Bussler and M. Matthew, *The Semantic Web: Semantics for Data and Services on the Web*, Springer, 2008.
- [4] R. Liepins, K. Cerans und A. Sprogi, „Visualizing and Editing Ontology Fragments with OWLGrEd,“ in *I-SEMANTICS 2012*, 2012, pp. 22-25.
- [5] H. Knublauch, R. Fergerson, N. Noy and M. Musen, „The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications,“ in *Third International Semantic Web Conference*, Hiroshima, 2004.
- [6] D. Bold, „Kompakte und ganzheitliche Visualisierung von Ontologien,“ MSc Thesis, Stuttgart University, 2013.
- [7] A. Jurcik, „Development of Visualization Plug-in for Protégé,“ MSc Thesis, Brno University, 2010.
- [8] S. Krivov, F. Villa, R. Williams, X. Wu. On visualization of OWL ontologies. In *Semantic Web*, pages 205-221. Springer, 2007.
- [9] S. Lohmann, S. Negru, D. Bold, „The ProtégéVOWL Plugin: Ontology Visualization for Everyone“, in „The Semantic Web: ESWC 2014 Satellite Events“, pages 395-400. Springer, 2014.
- [10] S. Lohmann, S. Negru, F. Haag and T. Ertl, „VOWL 2: User-Oriented Visualization of Ontologies“, in “Knowledge Engineering and Knowledge Management: 19th International Conference, EKAW 2014, Linköping, Sweden, November 24-28, 2014”, pages 266-281. Springer, 2014.