OntoCASE4G-OWL: Towards a modeling software tool for G-OWL a visual syntax for RDF/RDFS/OWL2

Michel Héon[1], Roger Nkambou[1], Mohamed Gaha[2]

[1]Université du Québec à Montréal, Montréal, Canada
heon@cotechnoe.com, nkambou.roger@uqam.ca
[2]Institut de Recherche en Électricité d'Hydro-Québec
Gaha.Mohamed@ireq.ca

Abstract. The W3C Web Ontology Language (OWL) provides the needed expressiveness formulation of complex concepts. However, the codification of an ontology is a thought formalization process that sometimes requires extensive knowledge and is often inaccessible laypersons. The G-OWL (for Graphical OWL) syntax has been designed to make easier the knowledge expression (compliant to OWL) in a graphical way. This paper presents the OntoCASE4G-OWL prototype, software for editing and modeling formal graphical ontology in G-OWL.

Keywords: OWL2, ontology, visual ontology modeling, visual modeling tool, visual knowledge representation

1 Introduction

With respect to the knowledge representation in an ontology, W3C offers five readable syntaxes that range from machine readable to human readable (RDF/XML, OWL/XML, Turtle, etc.). All of these notations operate in a text mode. In addition to being a tool for resources description, RDF/RDFS and OWL are usually used as domain knowledge representation language. Most aspects of the G-OWL design are grounded in cognitive science theory and based on years of research in knowledge engineering. Indeed, studies in cognitive science [1, 2] have shown some effectiveness (in terms of expression simplicity) when a visual notation is used for the ontological knowledge representation.

As an instance, Graffoo[1], Graphol[2] and VOWL[3] offer tools to visualize ontologies. In the same way, our research on G-OWL [3,4] proposes a visual modeling syntax for ontology design. G-OWL provides polymorphic and typological constructors [1] in order to minimize the number of symbols in the syntax. An important factor distinguishing G-OWL from other graphical syntaxes is that its design focuses on the OWL expressivity representation elements (hierarchy of concepts and roles, restriction, level of abstraction, etc.) and not on the strict visual representation of the syntactic

[1] http://www.essepuntato.it/graffoo
[2] http://www.dis.uniroma1.it/~graphol/
[3] http://vowl.visualdataweb.org/

OWL elements as proposed in the most recent research. The case ❹ describes below illustrates this design principle by the choice of representing Boolean predicates, not by a "relation" as it is often the case, but rather by a visual "entity" (which is a container in this case).

This paper presents OntoCASE4G-OWL[4], a prototyped software for G-OWL-based ontology editing and modeling. The prototype that will be presented in this Demo implements a subset of OWL expressivity and aims mainly to validate 1) the structural principles of the G-OWL syntax, 2) the technological architecture that supports the implementation of G–OWL and 3) the serialization/deserialization mechanism which ensures the ontology translation from Turtle to G-OWL and *vice-versa*. A subset of the `wine.owl` ontology will be used to illustrate the modeling capability of OntoCASE4G-OWL as well as its expressiveness (in terms of numbers of knowledge element types that it uses, including complex concept, property taxonomy, restriction, datatype, factual and conceptual statement, etc.).

2 Implementation of the G-OWL Syntax

As previously mentioned, G-OWL uses polymorphism and typology in order to limit the number of symbols necessary to model an ontology; hence, we exploit the assumption that using a limited number of symbols while preserving the expressive power facilitates the understanding of a syntax.

2.1 Hypothesis

For the design of G-OWL, our main hypothesis is that, it is possible to reduce the number of symbols required to design a G-OWL model while retaining OWL expressivity. To do so, it is necessary to apply polysemy on G-OWL symbols. Polysemy is the association of a finite number of meanings to a sign. Two techniques allowed us to increase G-OWL's polysemy:

1. **Polymorphism** is the expression of an object in several forms. It allows the attribution of configuration symbols on meaning. The meaning can later be disambiguated via its topological usage context. For example, in ontological modeling, the expression of a hierarchy can take two forms, either the class hierarchy or the properties hierarchy. In G-OWL, the expression of the hierarchy is symbolized by a `Slink`. The disambiguation of `Slink` is guided by the topology for the use of the links. It could result in `rdfs:subClassOf` if the `Slink` is used to link two classes or in `rdfs:subPropertyOf` if the `Slink` links two properties. In this example, the polymorphism restricts the symbolism of semantics that has only one symbol.
2. **Typology** assignment is the technique that limits the number of symbols by assigning them with types. Moreover, each type has a limited number of meanings. During modeling, this kind of typing has the advantage of clearly distinguishing the semantics associated with the language from the domain-dependant semantics as-

[4] http://www.cotechnoe.com/ontoase4gowl

sociated to the model. At the moment of formalization, the type is used as a guide in the disambiguation process.

2.2 User Interface

OntoCASE4G-OWL is an Eclipse-based application that operates: the Sirius graphical Framework for the diagram interface implementation, the Eclipse Modeling Framework (EMF) for data management of the models and Apache Jena for the Turtle - G-OWL de/serialization. With a Eclipse plug-in architecture, OntoCASE4G-OWL can be integrated (and is compatible) with other Eclipse software engineering tools, such as TopBraid Composer® or IBM Rational Software Architect® thus offering to the ontological engineer an additional functionality to facilitate knowledge elicitation with the contents experts.

Figure 1 presents a user interface input screen of OntoCASE4G-OWL. The arrow indicates the icons to create either a modeling project, or a G-OWL model. Five views are presented to the user. View ① presents the tree of the ontology structure. In ② is presented the schematized view accompanied by the palette to create ontological elements in G-OWL ontology. In ③ is presented a summary of the schema facilitating navigation in a broad diagram. ④ presents the properties associated with the selected item., The interface ⑤ presents the Turtle code interpretation of the G-OWL model. Finally, ⑥ presents the canvas palette of G-OWL.

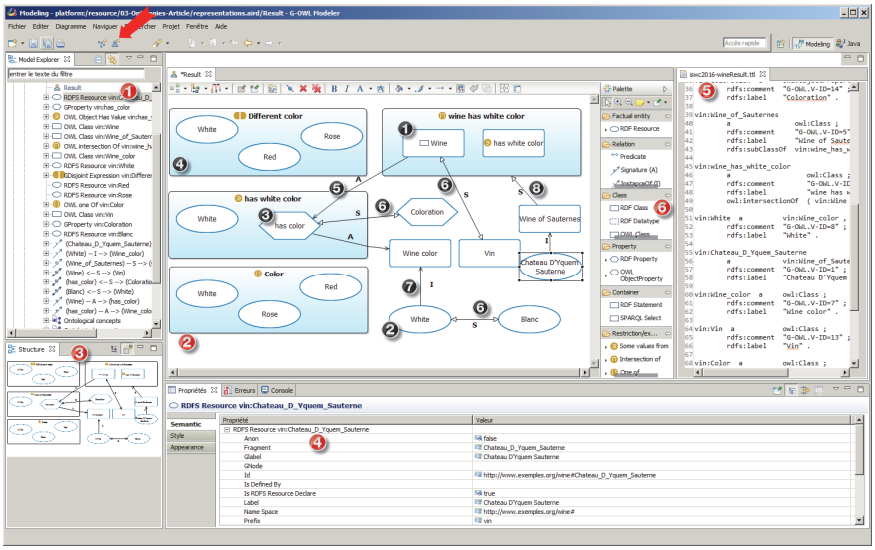

Fig. 1. Visual environment of OntoCASE4G-OWL

The G-OWL model in ② describes an ontology inspired by `wine.owl`. The element in ❶ represents an `owl:Class`, in ❷ it represents a `rdfs:resource` and in ❸ it represents an `owl:ObjectProperty`. In ❹, the model present 4 typed elements containers, that is: an `owl:AllDifferent`, `owl:intersectionOf` boolean

expression, an `owl:hasValue` restriction and an `owl:oneOf` expression. It is to be noted that these four graphic elements are built regardless of the fact that their semantics is associated with an ontological entity or an ontological predicate. This is an important characteristic of G-OWL. Indeed, the choice of the syntactic element shape is determined by its employment (i.e. in this case a container) rather than by its nature in a triplet (a subject/object or a predicate). Further, in ❺ the polymorphism of the `Alink` is disambiguated by `rdf:domain` of `rdf:range` depending on the arrow orientation. For ❻, the equivalent link (`DSLink`) polymorphism is disambiguated by the usage context: `owl:sameAs` between two `rdfs:Resource`, `owl:equivalentProperty` between two `owl:ObjectProperty` and finally by `owl:equivalentClass` between two `owl:Class`. For typed links in the ❼ - ❽ those are disambiguated in `rdf:type` and `rdfs:subClassOf`.

2.3 G-OWL/Turtle serialization/deserialization

The Turtle serialization/deserialization module ensures the translation of G-OWL notations to Turtle and *vice-versa* in accordance with the vocabulary described in [3, 4]. In the translation process, the module applies a set of rules used to disambiguate the G-OWL model. OntoCASE4G-OWL is built to perform a "hot" (dese/se)rialization without any particular import/export actions from the user.

3 Conclusion

Preliminary validation results of G-OWL was already presented in [4] and a thorough empirical validation with users are currently under development. OntoCASE4G-OWL is now used in two projects: 1) an ontology engineering project for the design of an expert knowledge base of electrical units, and 2) a project aiming at capturing data semantics representation in Big-Data. Works are currently underway to extend the OntoCASE4G-OWL functionality to the whole OWL-2 expressivity. The usage of polymorphism and typology are greatly exploited to express the various OWL quantifiers in G-OWL.

Acknowledgment
We wish to thank Hydro-Québec and the Canadian MITACS Scholarship Program funds for the financing of this research.

Reference

1. Paquette, G., Graphical Ontology Modeling Language for Learning Environments. Technology, Instruction, Cognition & Learning, 2007. 5(2): p. 36..
2. Basque, J., et al., Collaborative Knowledge Modelling with a Graphical Knowledge Representation Tool: A Strategy to Support the Transfer of Expertise in Organisations, in Knowledge Cartography: Software Tools and Mapping Techniques, A. Okada, S.B. Shum, and T. Sherborne, Editors. 2008, Springer London: London. p. 357-382.
3. M. Héon, R. Nkambou, and C. Langheit, "Toward G-OWL: A Graphical, Polymorphic And Typed Syntax For Building Formal OWL2 Ontologies," presented at the Proceedings of the 25th International Conference Companion on WWW, Montréal, Canada, 2016.
4. M. Héon, Web sémantique et modélisation ontologique (avec G-OWL): Guide du développeur Java sous Eclipse, Collection Epsilon ed.: Editions ENI, 2014.