Towards a UML-based notation for OWL ontologies

María Poveda-Villalón[1,*], Serge Chávez-Feria[1], Sergio Carulli-Pérez[1] and Raúl García-Castro[1]

[1]*Universidad Politécnica de Madrid, Madrid, Spain*

Abstract

Ontology conceptualization is a crucial task within the ontology development process. During this activity, developers should generate a conceptual model of the ontology based on the requirements that the ontology should meet. Very often, this activity is carried out by producing diagrams to represent the ontology structure and main elements; to do this, paper, blackboard, or digital drawing tools could be used. From this point on, the generated models drive the implementation of the ontology. Therefore, the closer the diagrams are to the OWL constructs, the easier the ontology developer would be to generate the ontology code. However, up to now, there is no standard notation for OWL that reduces the freedom of modelling between custom diagramming styles and the OWL code. For this reason, in this paper we introduce the Chowlk notation for the OWL language based on UML. This notation is being adopted in several projects including ontologies developed by standardization bodies.

Keywords

Ontology conceptualization, Ontology notation, Knowledge representation

1. Introduction

According to the NeOn methodology “Ontology Conceptualization” refers to: *the activity of organizing and structuring the information (data, knowledge, etc.), obtained during the acquisition process, into meaningful models at the knowledge level according to the ontology specification document. This activity is independent of the way in which the implementation of the ontology will be carried out.* (definition taken literally from [1]). For carrying out this activity, even though it is not the only solution, ontology developers often use graphical representations as it is more convenient to provide an overall idea of the concepts and relations involved in the model, and it is a powerful tool to communicate the model to potential users or domain experts.

The NeOn definition notes that the conceptualization is independent of the ontology implementation. However, it is advisable to use a conceptualization mechanism as close as possible to the ontology implementation language in order to avoid ambiguity, reducing therefore the ontology developers’ effort when translating the conceptualization to the actual code. For this reason, as a result of the fact that there is no standard notation for graphically representing ontologies, in this paper a UML based notation for ontology conceptualization targeting OWL is presented in Section 2. This notation is compared with the existing efforts in Section 3.

VOILA23: Visualization and Interaction for Ontologies, Linked Data and Knowledge Graphs
mpoveda@fi.upm.es (M. Poveda-Villalón); serge.chavez.feria@upm.es (S. Chávez-Feria); sergiomario.carulli.perez@upm.es (S. Carulli-Pérez); rgarcia@fi.upm.es (R. García-Castro)
0000-0003-3587-0367 (M. Poveda-Villalón); 0000-0002-7454-9202 (S. Chávez-Feria); 0000-0002-0421-452X (R. García-Castro)

2. The Chowlk Visual notation

The Chowlk visual notation[1] presented in this paper is based on the UML_Ont profile [2]. It should be mentioned that while the original UML_Ont profile utilizes custom stereotypes and dependencies to cover OWL 1, the Chowlk notation binds the stereotypes used in the profile to OWL, RDF(S) constructs and some OWL 2 constructs. Another reason for developing the Chowlk notation is to provide more compact alternatives to represent property characteristics and axioms than the UML_Ont profile. A preliminary version of the notation was presented in [3] as guidelines for ontology documentation. The following sections detail the diagram elements used to represent OWL ontologies metadata, concepts, properties, and individuals.

2.1. Basic Elements

Figure 1 shows the basic elements of the Chowlk visual notation. First of all, to represent the ontology and its elements URIs the notation should allow the definition of namespaces to make more compact representations. In addition, namespaces and prefixes should be declared in the conceptualization in order to avoid ambiguity in the declaration of the elements, for example, to use two concepts from different ontologies but with the same identifier. A specific block is included in the notation for the declaration of these namespaces (Figure 1 (a)).

Another specific block is defined in the notation to allow the declaration of ontology metadata (Figure 1 (b)) by listing the annotation properties needed and their values for the given ontology. Annotation properties should have the prefix and name of the property. The prefixes of the properties should also be defined in the Namespaces declaration block mentioned above. This block is also used to declare `owl:imports`.

Figure 1 (c) to (h) represent the rest of the basic visual elements used by the Chowlk notation. These main elements constitute the basis of an ontology conceptualization and have one or more OWL counterparts, that is, each element may be translated into different OWL elements depending on how the elements is used in the conceptualization. The possible OWL elements represented are shown in Figure 1 under each shape. For example, classes and individuals can be represented by rectangles (Figure 1 (c)). In this case, if the URI of the element is underlined, it represents an individual, and it would be a class if not underlined, as will be explained later. This shape, the rectangle, could be also used to represent datatype properties.

When referring to RDF/RDFS/OWL constructs URIs, the double angled brackets (« ») are used around the specific RDF/RDFS/OWL URIs included in the diagrams.

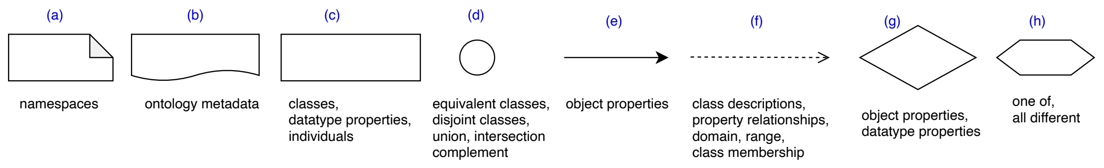

Figure 1: Chowlk visual notation basic elements.

[1]More detailed information about the notation is provided at https://chowlk.linkeddata.es/notation.html.

2.2. Classes

Named classes are represented by a rectangle that includes the class URI (Figure 2 (a)). "Subclass of" is represented by an empty arrow (Figure 2 (b.1)) or using the `rdfs:subClassOf` stereotype within a dashed arrow (Figure 2 (b.2)). Anonymous classes are defined by an empty rectangle (Figure 2 (h.2)) or circles representing `owl:intersectionOf` or `owl:unionOf` (Figure 2 (c) and (d)). Equivalence and disjointness between classes are represented by circles as shown in Figure 2 (e) and (f). The circles with the logic operators inside them represent the axioms, meanwhile the classes connected to those circle using arrows are the concepts involved in the relationship. Classes enumerations are defined using the hexagon linking to the involved individuals (Figure 2 (g)).

Classes defined using property restrictions can be represented concisely by including the operators used in the arrows representing the object properties (Figure 2 (h.1), (j), (l)) or before the datatype properties (Figure 2 (i), (k), (m)). As shown in Figure 2, the tags (all) and (some) indicate existential and universal restrictions, respectively. The tag (N1..N2) represents cardinality. If the user wants to specify an exact cardinality, the numbers N1 and N2 should be equal. The notation has the "N" letter reserved for the cases in which no maximum cardinality is needed. For instance, the tag (2..N) will indicate a minimum cardinality restriction of 2, and a nonexistent maximum cardinality restriction.

As the notation used in Figure 2 (h.1), (j) and (l) for object properties (and (i) (k) and (m) for datatype properties) could be combined with subclass of and equivalent class axioms, it should be defined to which OWL construct it would correspond. The decision was to understand these options as "subclass of" because the logical commitment is less strong than an equivalent class in terms of reasoning and because it is more likely to find that pattern in existing ontologies. In order to allow for having equivalent classes with the universal, existential and cardinality constraints, the tag "(eq)" is added before the constraint, as shown in Figure 2 (n.1), (o) and (p). It should be mentioned that options (h.1) represent the same modelling as (h.2) and the same holds between (n.1) and (n.2).

2.3. Object properties

Object properties are represented by labeled directed arrows where the head of the arrow indicates the possible domain and range of the property. Several variations can be applied to the style of the arrow depending on their individual particularities. Figure 3 options (a.1) to (d.2), provides a summary of the possible combinations. More precisely, if the property domain is the attached class, the dot at the origin of the arrow is filled (Figure 3 (a.2) and (b)), otherwise it is empty (Figure 3 (c) and (d.2)). If the property range is the attached class, the arrow is solid (Figure 3 (a.2) and (c)), otherwise it is dashed (Figure 3 (b) and (d.2)). To simplify the cases where both the domain and the range are defined, the plain arrow without a circle at the origin could be used as a shortcut (Figure 3 (a.1)) and the same could be used for cases where neither the domain nor the range are defined using the dashed arrow (Figure 3 (d.1)). This notation feature is designed to allow the developer to state between which classes the property intended is to be used, even though there is no domain and range declared. This graphical information is very valuable to a potential user when starting reading an ontology as it is not explicitly declared in

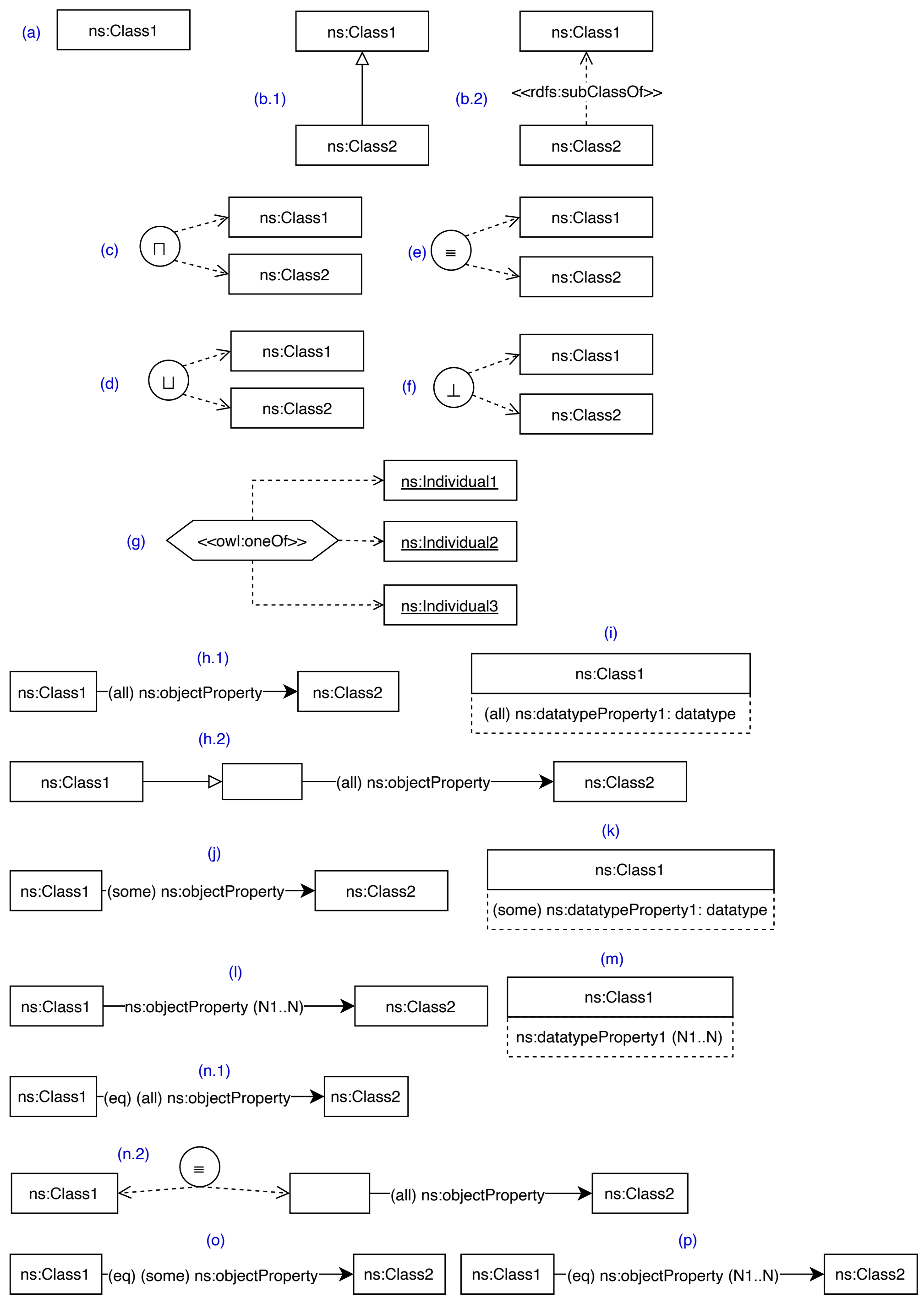

Figure 2: Class definitions, axioms and constraints.

the code.

The property characteristics are also indicated by special tags in the labels. The tags (F), (S), (IF), (T) indicate if a property is `owl:FunctionalProperty` (e), `owl:SymmetricProperty` (f), `owl:InverseFunctionalProperty` (g) or `owl:TransitiveProperty` (h) respectively. One of the advantages of using tags to declare things like property restrictions (Figure 2 (h.1) to (n.2)) and property types is that they do not interfere with other definitions of properties, such as the domain and range, which control the style of the arrow. The inclusion of concise tags avoids the use of additional elements that will clutter the conceptualization.

The notation also provides an alternative representation for properties using the diamond block (see Figure 1). This shape, reused from UML_Ont profile [2], supports several use cases, especially to declare relationships between properties such as: `rdfs:subPropertyOf`, `owl:inverseOf`, or `owl:equivalentProperty` to avoid cluttering the main diagram. However, this could also be defined using the arrow notation as shown in Figure 3 (i) to (l).

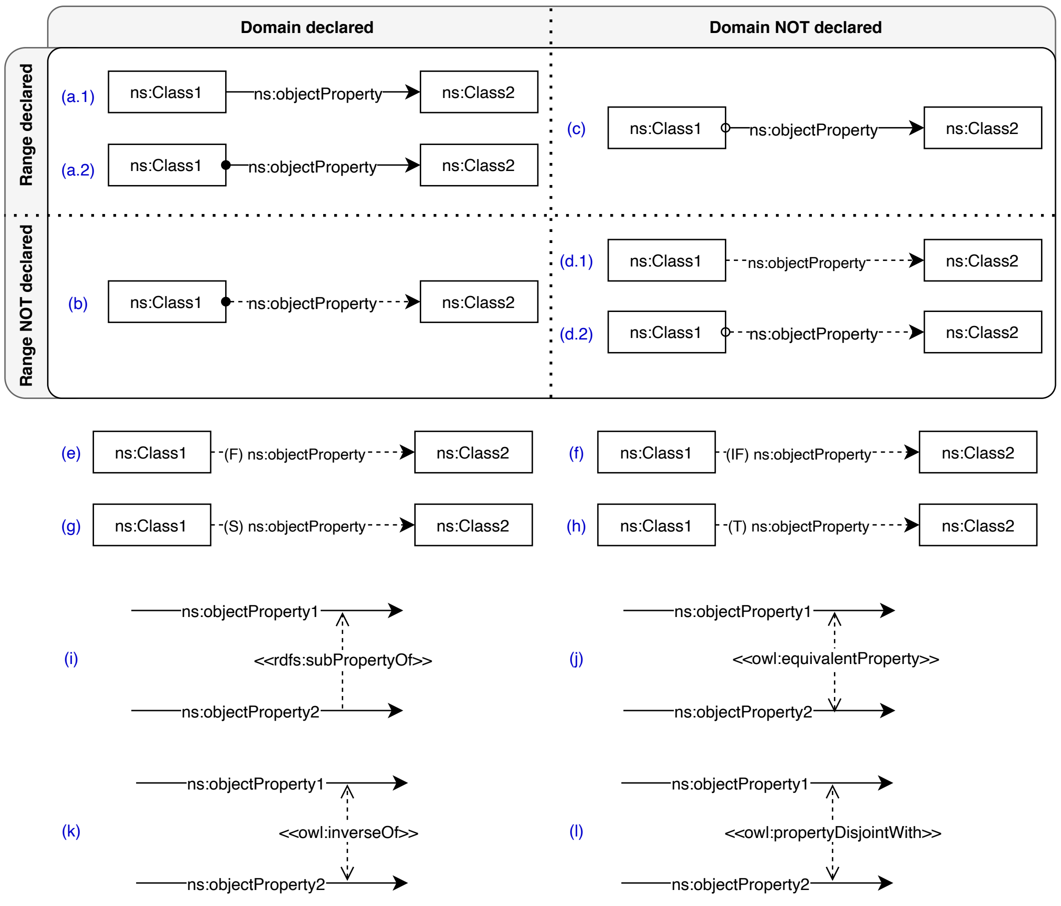
Domain declared
Domain NOT declared
Range declared
Range NOT declared
(a.1) ns:Class1 ns:objectProperty ns:Class2
(a.2) ns:Class1 ns:objectProperty ns:Class2
(c) ns:Class1 ns:objectProperty ns:Class2
(b) ns:Class1 ns:objectProperty ns:Class2
(d.1) ns:Class1 ns:objectProperty ns:Class2
(d.2) ns:Class1 ns:objectProperty ns:Class2
(e) ns:Class1 (F) ns:objectProperty ns:Class2
(f) ns:Class1 (IF) ns:objectProperty ns:Class2
(g) ns:Class1 (S) ns:objectProperty ns:Class2
(h) ns:Class1 (T) ns:objectProperty ns:Class2
(i) ns:objectProperty1 <<rdfs:subPropertyOf>> ns:objectProperty2
(j) ns:objectProperty1 <<owl:equivalentProperty>> ns:objectProperty2
(k) ns:objectProperty1 <<owl:inverseOf>> ns:objectProperty2
(l) ns:objectProperty1 <<owl:propertyDisjointWith>> ns:objectProperty2

Figure 3: Object property definitions, characteristics and relations.

2.4. Datatype properties

Visually, datatype properties or attributes are shown as complementary rectangles below classes, as shown in Figure 4. A rectangle can contain more than one attribute. As in the case of object properties, the user can modify the style and text inside those rectangles in order to represent whether domain and ranges are defined. A solid border line of the rectangle indicates that the domain of that set of attributes is the class defined above them (Figure 4 (a) and (b)). A dashed line indicates that the attribute is intended to be used for instances of the attached class; however, the domain is not declared (Figure 4 (c) and (d)). The range is indicated by declaring a datatype in the text of the attribute (Figure 4 (a) and (c)).

Restrictions over the attributes are also possible using the same tags explained for the object properties in and shown in Figure 2 (i) (universal), (k) (existential) and (m) (cardinality). In the case of property types, only the functional tag (F) applies to attributes, according to the OWL specification. If a datatype is functional, it is defined using a tag as shown in Figure 4 (e).

The notation also allows representing the datatype relations using the diamond block to define `rdfs:subPropertyOf` (Figure 4 (f)), `owl:equivalentProperty` (Figure 4 (g)) and `owl:propertyDisjointWith` (Figure 4 (h)).

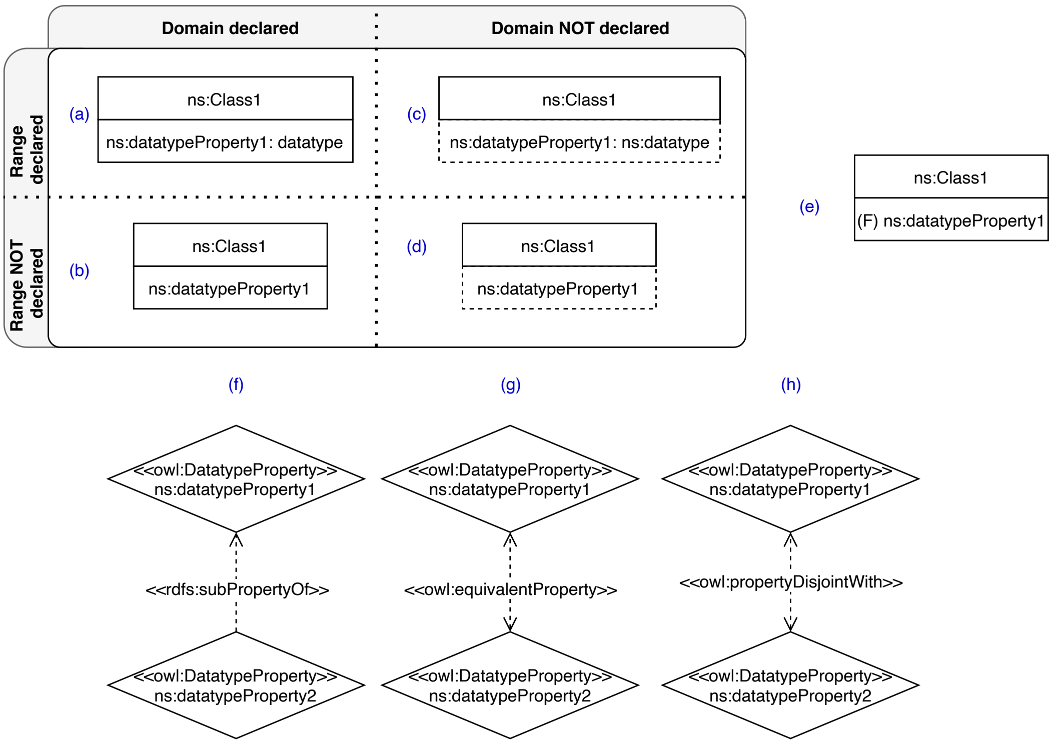

Figure 4: Datatype property definitions, characteristics and relations.

2.5. Individuals

The Chowlk notation uses the same rectangular visual block to represent classes and individuals; however, the individuals have their labels underlined (Figure 5 (a)). Figure 5 shows four options to state the class membership of an individual (b.1 to b.4)). The first option (b.1) where the individual is attached to the class it belongs to is the preferred one, as it is more compact and reduces the number of arrows. If an individual belongs to more than one class, (b.1) and (b.3) are recommended.

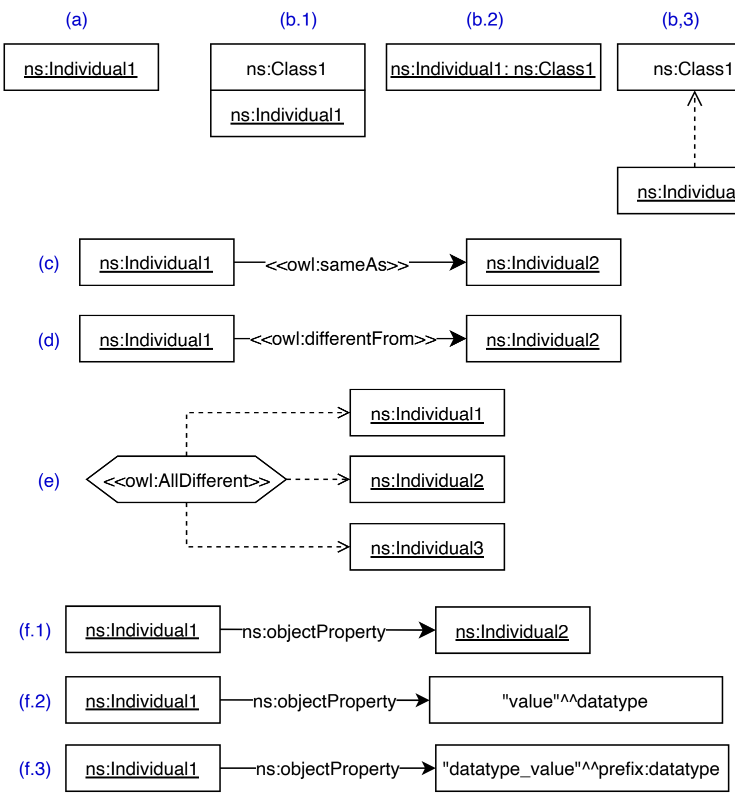

Figure 5: Individual definitions and use.

The notation also allows for the definition of individuals' identity axioms. Two equal individuals can be represented by a solid arrow connecting them with the `owl:sameAs` construct (Figure 5 (c)) and to state that two individuals cannot represent the same entity the `owl:differentFrom` construct can be declared (Figure 5 (d)). To indicate that a set of individuals are all different, the hexagonal visual block shown in Figure 5 (e) could be used. In this case, the hexagonal label contains the axiom `owl:AllDifferent` and is connected to all individuals involved in the axiom using dashed arrows.

It is also possible to instantiate the property values as shown in Figure 5. In the case of object properties, the individuals are connected by arrows stating the relation in the label (f.1). In the case of datatype properties, the same artifact is used, but now the destination of the arrow is connected to a datatype value (f.2) and (f.3).

3. Related work

During the last decades, ontology visualization and graphical edition have attracted researchers attention. For example, the systematic literature review provided by [4] in which the relation between UML and ontology based formalisms is investigated through the analysis of 66 works. In addition, a comprehensive analysis of visualization methods and tools and their capabilities is provided by [5]. However, in this paper we focus on the notation aspect independently of the technology transforming the conceptualization into code.

In this sense, we can compare our work to existing notations which could be broadly classified in directed graphs oriented or UML alike. One of the well-known of the former type is WebVOWL [6]. In this case, the type of notation used makes it more complex to state a number of object properties between the same two classes. It also generates less compact diagrams as the datatype properties are also arcs to new nodes (therefore, more nodes are needed than in UML-alike visualizations). Finally, the hierarchies cannot be placed in a tree-oriented way as the visualization approach follows a circular shape. A closer notation to UML is the Graffoo specification. In this case, the issue with more cluttered diagrams because datatype properties are also represented with nodes remains. In addition, the class constraints are also represented by an additional box, in contrast to reusing the object properties arrows as proposed by the Chowlk notation, generating cleaner diagrams.

Both WebVOWL and Graffoo use colours in the shapes to convey information about the model, which can be lost, for example, when printing in black and white or for people with different colour perception. In the case of the Chowlk notation, the colour of the classes are meaningless from the conceptualization point of view but it is advisable to use different colours for different namespaces so that the user can easily identify in which ontology is every concept defined. See, for example, the SAREF4CITY ontology.[2]

Another notation following UML is OWLGrEd [7]. However, from its notation description[3] it is not clear which OWL features are supported, for example imports, class equivalence, property equivalences, cardinality constraints, etc.

4. Conclusions and future work

This paper has presented foremost characteristics of the Chowlk notation and major advantages over existing ones. Also, the main rationales for some notation characteristics, such as domain and ranges options and the use of tags for property characteristics and class constraints, have been explained. This notation is provided within the Chowlk framework and two diagrams.net libraries are available containing a complete version or a subset of the notation elements.[4]

[2]https://saref.etsi.org/saref4city/
[3]http://owlgred.lumii.lv/notation
[4]The lightweight and complete libraries are available at https://chowlk.linkeddata.es/notation.html

The notation has been used in a number of projects including ontologies developed by standarization bodies as, for example, the ETSI SAREF ontologies[5] as it can be seen in the SAREF4AGRI ontology[6] among others. Furthermore, by W3C communities, such as the RML ontologies[7] and the WoT discovery ontology.[8] Other examples are European projects as BIMERR,[9] and COGITO.[10] It is being adopted by projects carried out by organizations not related to the authors of this paper as BIM4EEB[11], CosWOT [12], D2KAB[13], and Mat-o-Lab.[14]

The main future line of work involves the detailed comparison with existing notations in terms of capabilities, clarity, and ability to generate compact diagrams. The notation documentation is planned to be improved overall incorporating more examples. The generation of examples for well-known ontologies is part of the Chowlk roadmap in order to ease their reuse.

Finally, as the notation is used by the Chowlk converter to generate OWL code, it is needed to define notation characteristics to indicate the converter from where to start reading axioms in order to reduce the complexity when processing the notation underlying graph. However, this need is oriented to the technological side of the framework rather than to the visualization capabilities to represent OWL ontologies.

Acknowledgments

This work has been supported by COGITO funded by the European Union's Horizon 2020 research and innovation programme under grant agreement no. 958310 and by the Madrid Government (Comunidad de Madrid-Spain) under the Multiannual Agreement with the Universidad Politécnica de Madrid in the Excellence Programme for University Teaching Staff, in the context of the V PRICIT (Regional Programme of Research and Technological Innovation).

References

[1] M. C. Suarez-Figueroa, A. Gómez-Pérez, First attempt towards a standard glossary of ontology engineering terminology (2008).
[2] P. Haase, S. Brockmans, R. Palma, J. Euzenat, M. d'Aquin, D1.1.2 updated version of the networked ontology model, Technical Report, Universität Karlsruhe, 2009. NeOn Project. http://www. neon-project. org.
[3] D. Garijo, M. Poveda-Villalón, Best practices for implementing FAIR vocabularies and ontologies on the Web, 2020. doi:10.3233/SSW200034.

[5]https://saref.etsi.org/extensions.html
[6]https://saref.etsi.org/saref4agri/
[7]https://kg-construct.github.io/rml-resources/portal/
[8]https://github.com/w3c/wot-discovery/tree/main/context
[9]https://bimerr.iot.linkeddata.es/
[10]https://cogito.iot.linkeddata.es/
[11]https://digitalconstruction.github.io/v/0.5/index.html
[12]https://coswot.gitlab.io/
[13]http://www.elzeard.co/ontologies/c3po/plant#
[14]https://github.com/Mat-O-Lab/MSEO

[4] M. Mejhed Mkhinini, O. Labbani-Narsis, C. Nicolle, Combining uml and ontology: An exploratory survey, Computer Science Review 35 (2020) 100223. doi:`https://doi.org/10.1016/j.cosrev.2019.100223`.

[5] M. Dudás, S. Lohmann, V. Svátek, D. Pavlov, Ontology visualization methods and tools: a survey of the state of the art, The Knowledge Engineering Review (2018) 33.

[6] V. Wiens, S. Lohmann, S. Auer, Webvowl editor: Device-independent visual ontology modeling, International Semantic Web Conference (2018).

[7] J. Barzdins, G. Barzdins, K. Cerans, R. Liepins, A. Sprogis, UML style graphical notation and editor for OWL 2, in: Perspectives in Business Informatics Research - 9th International Conference, BIR 2010, Rostock Germany, September 29-October 1, 2010. Proceedings, volume 64 of *Lecture Notes in Business Information Processing*, Springer, 2010, pp. 102–114. doi:`10.1007/978-3-642-16101-8\_9`.