Designing and Communicating Ontologies Visually

Gilbert Paquette[1] [0000-0002-2898-3462] and Michel Héon[2] [0000-0001-7515-6382]
[1] LICEF Research Institute, Université TELUQ, Montreal, Canada, gilbert.paquette@licef.ca
[2] Cotechnoe Inc and UQAM Université du Québec à Montréal, Montréal, Canada, heon@cotechnoe.com

Abstract. In this paper we discuss ways to support the Ontology Engineering process by providing a Visual Language for OWL 2 ontologies. We examine eight proposals for an ontology visual language stemming from Semantic Web research, including our own Visual Ontology Language that has evolved from our MOT semi-formal visual language. The MOT-OWL visual language implements the visual typing of ontology entities and links, and also the use of polysemy between these elements to increase the readability and manageability of the visual models. Then we compare this visual notation and other Visual Ontology Languages using principles from the Physics of Notation Theory. This comparison helps us identify improvements that are being implemented in our more recent visual language, G-OWL, based on a systematic meta-modeling effort.

Keywords: Ontology, OWL-2, Knowledge Representation, Visual Ontology Language, Ontology Engineering, Ontology Readability.

1. Introduction – Ontology Languages

Ontology languages are central for the implementation of the Semantic Web, a field of information technology in which Symbolic Artificial Intelligence and Web Technologies converge. The Semantic Web is based on shared formal knowledge representations that can evolve and on software agents that can manipulate these representations. "For the semantic web to function, computers must have access to structured collections of information and sets of inference rules that they can use to conduct automated reasoning." [2]. Thus, a central purpose of the Semantic Web is to introduce explicit descriptions about the meaning of web resources, to increase computer understanding of the Web's content.

The representation of knowledge in the Semantic Web is based on the concept of an ontology. As defined by the OMG [27], "*An ontology defines the common terms and concepts (meaning) used to describe and represent an area of knowledge. An ontology can range in expressivity from a simple Taxonomy (knowledge with minimal hierarchy or a parent/child structure), to a Thesaurus (words and synonyms), to a Conceptual Model with more complex knowledge, to a Logical Theory with very rich, complex, consistent, and meaningful knowledge".* More expressive knowledge representation enables more sophisticated reasoning capabilities and more intelligent behaviors of information agents, humans and machines. The Web Ontology Language (OWL), on which we will focus in this article, is at a high level of expressivity, thus providing a rich reasoning capability without risking the applications to become uncomputable if higher level languages were used. The different dialects of OWL are based on Description Logics, which are subsets of First Order Logic (FOL). OWL ontologies are schemas that can process *Resource Description Framework /Schema (*RDF/S) triples. By combining such triples, the instances of the ontology can be envisioned as a Knowledge graph that represents a domain of knowledge (meaning and fact).

First, we have to make an important distinction between two main orientations in the use of visual ontologies. The first approach aims at the visual representation of large sets of data to uncover a set classes organisation and some semantics to organize the data. The second one is dedicated to model extensively an area of knowledge providing its semantics and data organization, which can be used to generate new data sets. While both approaches are complementary, the choice of a focus will determine the kinds of visual symbols in the Visual Ontology Language. Our focus in this article is on the second top-down approach from the ontology to its data instantiation.

1.1. Ontology Engineering

Our goal in building Visual Ontology Modeling Languages is to support content expert and ontology designers involved in Ontological Engineering. This discipline groups a set of design principles, development processes, tools and systematic methods to facilitate ontology development and use. According to Mizoguchi and Kitamura [22], knowledge engineering for an intelligent system should always include ontology development tools and methodologies.

Several Ontology Engineering methodologies have been reported in the literature from surveys such as those in [6, 17, 35] propose a series of precise steps in the ontology-building process. The *Methontology framework* [11] is a pioneering comprehensive methodology that requires the definition and standardization of the entire ontology life cycle from the specification of the ontology requirements, the identification of the ontology development process based on evolving prototypes, the steps, techniques, products and evaluation procedures for each prototype and the final deployment and maintenance process of the ontology. Methontology recognizes the importance of knowledge acquisition, a long process working with domain experts, particularly important in the early phases of specification and conceptualization of the ontology.

Considering ontology development methodologies, one can understand the criticality of an efficient ontology development environment and the use of tools to help manage the various versions of the ontology, convert them in other formats and languages and evaluate and link ontologies from various sources.

Protégé is the leading ontology development editor and environment that supports most ontology engineering tasks and several ontology languages, such as OWL and RDF(S). Protégé [33], as well as other ontology development environments such as TopBraid [36] or Neon Toolkit [14] are basically form-based editors with a sophisticated user interface where the user can describe a class-subclass hierarchy in one view, then moves to other view to edit the classes and their associated properties. Properties can be described in still another view and in another one, assertions can be made or inferred by triggering a reasoning tool, for example Pellet or HermiT. The inference mechanisms integrated as plug-ins in Protégé facilitate the validation of the ontology and the discovery on unexpected discrepancies.

Building an ontology is a delicate and hard task and a global visual view of the knowledge graph with various specialized view is essential. Separating class, properties, and individuals in different forms blurs the necessary integrated view of an ontology. Some plug-ins have been added to Protégé to visualize ontologies, but they provide incomplete visualization support. A comparative survey of many other tools and environment for building ontologies [9] has also identified the fact that there is a lot of room for improvement in all these environments.

As Gasevic [12] pointed out: “Many users would like friendlier visual/spatial navigation among concept trees/graphs and relations, more options for using reasoning facilities to help explore, compose, and check ontologies, more features for aligning ontologies with one another, and tools that would help integrate ontologies with other data resources such as enterprise databases. The desirable improvements also include support for natural-language processing and collaborative development”.

The overall sentiment expressed by users of the various ontology development environments clearly reflect the need for facilitating the use of such ontology tools by domain experts and casual ontology users, not only by specialized ontologists. Providing a visual language and editor for designing OWL ontologies aims at his goal.

1.2. Textual and Visual Ontology Languages

We will now point out some of the attributes that distinguish a visual language from a textual language. First, let us underline that all the ontology standards adopted by World Wide Web Consortium (W3C) are textual languages. From the beginning, despite its RDF and RDFS graph basis, the main preoccupation of the W3C has been to enable machine readability for the use on ontologies in software applications. Thus far, the formal concrete syntaxes for OWL recommended by the W3C, such as OWL/XML, RDF/XML, Turtle, and the Functional or Manchester syntaxes are all text-based languages. The linear textual descriptions produced using these standards blurs the structure of the ontology and makes it difficult to design new ontologies. Sets of triples are sometimes represented using limited graphs for explanation purposes, but there is yet no W3C standard visual concrete syntax for OWL 2 ontology modeling

Larkin and Simon [19] note that the fundamental difference between a "graphic" or visual notation and a textual notation is that a visual notation explicitly presents the information on topological and geometric relationships between the components represented. Moody [23] underlines that visual notations also differ from the textual notation by the nature of the symbols that compose the vocabulary as well as by the rules governing the use and interpretation of symbols. In a textual notation the symbols are displayed following a one-dimensional (linear) layout, sequentially aligned to form words and words to form statements. The *unidimensional linearity* and the *sequential layout rules* are the two important notions that characterize a textual notation, hiding the structural nature of an ontology.

A visual notation uses visual symbols (geometric shapes, icons, pictograms, etc.) differentiated by a visual vocabulary using color, size and position of geometric shapes, together with rules for their visual arrangement, *surface* rules for a 2D representation or *space* rules for 3D representation. Hybrid notation (visual and textual) uses a vocabulary composed of both texts and visual symbols that are governed by rules of textual and visual arrangement.

Our research goal here is to provide to ontology designers and content experts a completely visual language environment that exports to W3C textual ontology languages standards such as RDF/XML or Turtle so they can be used in software applications and imported into ontology development environments such as Protégé for extension and validation operations.

1.3. Ontology Visual Languages Requirements

We now state essential requirements or principles for an ontology visual language to support ontology design and communication in the ontology engineering process. These requirements are based on the Physics of Notations Theory (PoNT) a systemic framework that has been used evaluate, compare, improve and design visual notations in a wide variety of fields, including Unified Modeling Language (UML) or Business Process Modeling Notation (BPMN) visual notations in software engineering [13,24,32].

PoNT [23] states nine principles as guidelines to design cognitively effective visual notations optimized for human communication and problem solving: semiotic clarity, perceptual discriminability, semantic transparency, complexity management, cognitive integration, visual expressiveness, dual coding, graphic economy, and cognitive fit. These nine principles were synthesized from theory and empirical evidence from a wide range of fields and rest on an explicit theory of visual communication. Here we separated or regrouped these principles (mentioned in parentheses) to account for the specificity of Visual Ontology Languages:

1 - Completeness: Each semantic OWL 2 object corresponds to a symbol or an understandable combination of symbols in the visual language; (part of *Semiotic Clarity*);

2 - Formality: Each visual symbol (or some combination of symbols) correspond to only one semantic OWL 2 object that can be disambiguated using the visual context; (part of *Semiotic Clarity*);

3 - Perceptual Clarity: Symbols should be clearly distinguishable from one another; notations should use the full range of the 7 visual variables: position, size, value, texture, color, orientation and shape; (grouping *Perceptual Discriminability* and *Visual Expressiveness*);

4 - Semantic Transparency: Notations should use graphical symbols whose appearance suggests meaning; graphical symbols can be evaluated from +1 to -1, from semantic transparent to opposite meaning; (rephrasing *Semantic Transparency*); the design of the visual notation is not a simple transposition of an OWL 2 text-based notation but reflect relations between its semantic features.

5 - Complexity Management: Notations should include explicit mechanisms for dealing with complexity, keeping the visual models at a reasonable size, favoring scalability for large ontologies, using sub-models if necessary or grouping symbols by types, integrating information between separate diagrams; *(*grouping *Complexity management* and *Cognitive Integration);*

6 - Totally visual: Notation should use text to complement (not replace) graphics; textual annotations should be used alongside graphics so that the notation remains totally visual; *(*rephrasing *no Dual Coding);*

7 - Parsimony/Polymorphism: The number of graphical symbols in the notation should be cognitively manageable; a large number of symbols increase complexity, thus reducing understanding; polymorphism is a way to reduce the number of visual symbols; (rephrasing *Graphic Economy*);

8 - Cognitive Fit: Different visual dialects should be used for different tasks and audiences; visual notations for OWL should be addressed primarily to content experts and ontology modelers; computer scientists will in general prefer textual representation of ontology constructs, using visual equivalents for overviews (rephrasing *Cognitive Fit*);

9 - Editing Tool for Computability: There must exist a tool that edits and translate a visual graph automatically to an OWL 2 standard text such as RDF-XML or Turtle so it can be used for computer scientists developing applications; (*particularity of OWL graphs*).

The visual ontology languages or notations to be presented in the following sections will be discussed according to these principles. In section2, we first present our own MOT-OWL ontology visual language and its GMOT-OWL editor. Sections 3 will discuss proposals for using the Unified Modeling Language (UML) to represent ontologies visually. Section 4 surveys five non UML proposals for a Visual Ontology Language, including our new G-OWL editor. In section 5, we proceed with a comparative analysis of all these proposals.

2. The MOT-OWL Ontology Visual Language

The MOT (Modeling with Object Types) Visual Language enables users to build a visual representation of knowledge from informal conceptual maps, to semi-formal knowledge graphs, up to RDFS or OWL-DL ontologies. Rooted in cognitive science, MOT has served in numerous Knowledge Management and Instructional Engineering projects in some large organizations for the last 20 years, thus providing experimental validation [29]. This experimental work provides a basis for the GMOT-OWL and G-OWL ontology visual languages to be discussed in this section and section 5.

The actual GMOT-OWL [30] visual editor has been specialized from the MOT previous editors to build RDFS and OWL-DL ontologies. The visual ontology models can be exported to W3C textual standard formats such as RDF-XML and Turtle, so they can be integrated in ontology development systems such as Protégé. GMOT-OWL ontologies have been used in the model-driven development of the TELOS platform [31] and also to build a Competency Ontology (Paquette et al., actually in publication)

2.1. Symbols of the MOT-OWL Visual Ontology Language

Figure 1 gives an overview of the visual vocabulary of the MOT-OWL language. The editor displays four kinds of graphic objects with different shapes and colors to represent classes (pink rectangles), properties (green hexagons) and individuals (blue rectangles with cut corners) plus icons for datatypes. These colors and the terms in the boxes are chosen by users. Special symbols are also used to represent data types, assert multiple disjoints or equivalents, combine classes or represent various property restrictions.

The use of various shapes and colors, and the names and direction of the links aim to satisfy the principle of *Perceptual clarity (3*). Although Object Property and Data Property symbols are the same, they can be distinguished by the Data symbol that serves as output of a Data Property, while a Class or Individual output signifies that it is an Object property. This is one way to implement the *Parsimony principle (7).* Another way to satisfy this principle is to use the same class or property symbols for subtypes with an annotation added. In the case of properties, as shown in lower right end of figure 3, a menu in the editor enables to add one to four symbols, **F, I, T** and **S**, to respectively state that the property is Functional, Inverse Functional, Transitive or Symmetric.

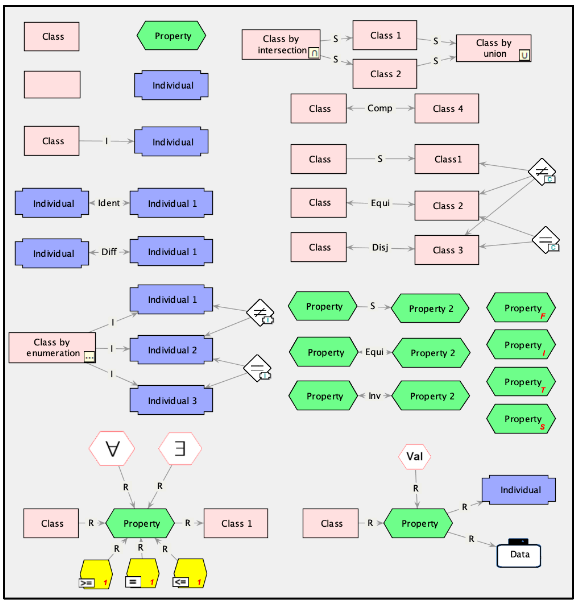


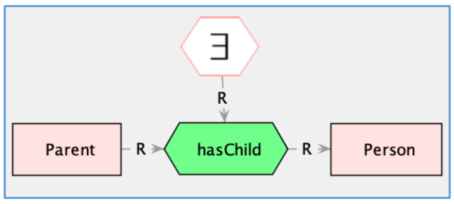
Fig. 1. The MOT-OWL visual vocabulary.

Classes can be declared simply by stating a name like "Class", as an anonymous class (with no name), as an enumeration of its members. Classes can also be declared as an intersection, union or complement of other classes.

One of the group of symbols presents the case of a class by enumeration using the *I instance* link (a fundamental of the general MOT language). The multiple Distinct link and the multiple Identical link applied to the three individuals are options. In the first case, it defines the class as a set with members that are *All Different*. In the other case, *Identical* individuals means that the class is a Bag where some or all of the individual are pairwise identical.

The use of the logic symbols ∀ (for all) and ∃ (exist), Boolean symbols ∪ (union) and ∩ (intersection) or cardinality symbols ≠, =, ≥ and ≤ are ways to provide some *Semantic Transparency (4)* since they refer to well-known mathematical or logic notions. The group of visual symbols on the lower left of figure 1 are restrictions *on Property* for class definition. They are to be used separately, one and only one with **R** ("ruled by") link to the property. The two upper symbols express *universal* or *existential* restriction on a property. The lower symbols express cardinality restrictions on the property, a minimum, exactly or a maximum of a an integer chosen by the user.

Figure 2 provides one example for a class defined by an existential restriction *owl:someValuesFrom*. This graph declares that a class named "Parent" groups members that have at least one value in class "Person" by the property "hasChild". The corresponding First Order Logic semantics and the Turtle translation are set alongside the visual MOT-OWL expression. The direction of the **R** links show "Parent" as a domain of "Property1" and "Person" as its range. This is another implementation of the *Semantic Transparency (4) principle.*



```
Parent owl:equivalentclass
  [a <owl:Restriction>;
     <owl:onProperty hasChild;
     <owl:someValuesFrom Person]
```

$(\forall x)\ (\text{Parent}\ (x) \equiv (\exists y)(\ \text{hasChild}\ (x\ ,\ y) \wedge \text{Person}\ (y)\ ))$

Fig. 2. A visual existential restriction with a Turtle and FOL translations.

Relations between classes, between properties or between individuals, use whenever possible the same links to respect the *Parsimony principle* (7) without affecting the *Perceptual Clarity (3)* of each notion that are disambiguated by their context. For example, the specialisation **S** link and the equivalence **Equi** links are the same for classes and properties, but they have of course different meanings that are disambiguated in the OWL-XML or Turtle translations by looking if they are set between Class symbols or between Property symbols.

2.2. Visual Language Requirements and the GMOT-OWL Editor.

We will now look at the other PoNT requirements besides principles 3, 4 and 7. We can first assert that GMOT-OWL is *Totally Visual* (6) since no text is used within the boxes except their identifiers or subtype marks such as "∪"or "…" for classes, or **F, I, R, S** for properties. Textual descriptions can be added for some visual objects in a companion window or by adding comments on the graph display, but they are complementary to the visual notation.

As for the *Cognitive Fit (8)* GMOT-OWL aims at a visual language for content expert or modelers, so a single profile of the editor was produced. We believe that computer scientists will prefer to use or develop ontology in one of the familiar textual formats like OWL-XML. These users will refer to the visual syntax from time to time to check overviews of their ongoing design. For this, the GMOT-OWL provides translation back and forth to the textual OWL standard (*Computability (9)*).

The first two principles of the introduction are part of the *Semiotic Clarity principle* in PoNT that requires a one-to-one correspondence between the visual symbols and their semantic meaning in the ontology. A more detailed presentation of GMOT-OWL [30] has demonstrated the *Completeness (1)* of GMOT-OWL: every OWL-DL object corresponds to a visual symbol (or a group of symbols). For some of the more complex semantic objects, such as restrictions on properties, class enumeration or multiple disjoints, a group of visual symbols is needed.

Conversely, not all visual symbols correspond to a semantic object. The visual notation in in overload, because the ∀ and ∃ symbols for example are not stand-alone semantic objects. But grouped with other visual signs as in figure 3, all the MOT-OWL symbols have clear semantic meaning.

2.3. Complexity Management in the GMOT-OWL Editor.

Complexity Management (5) aims to keep visual models at a reasonable size, to facilitate their understanding and their management. All the proposals for ontology visualisation mention the problem to represent large ontologies: soon the graph will look like a spaghetti of overcrossing links and overlapping objects too small to be well perceived cognitively.

The GMOT-OWL editor tries to avoid this situation by features like reducing the number of links, filtering a model to display only certain kinds of entities or links, and enabling decomposition of a large model into sub-models by copying entities with reference (as an alias) to avoid the overcrossing of links.

One example is given in figure 3 and 4 for a learning design ontology grouping a total of 38 classes, 47 properties, 101 individuals, 20 cardinality restrictions and 24 different relational links.

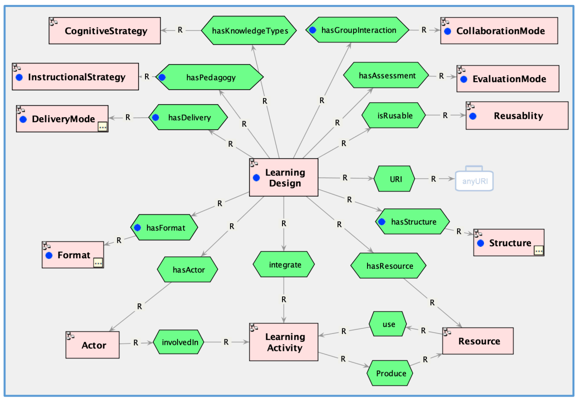


Fig. 3. A GMOT visual ontology for the Learning Design concept.

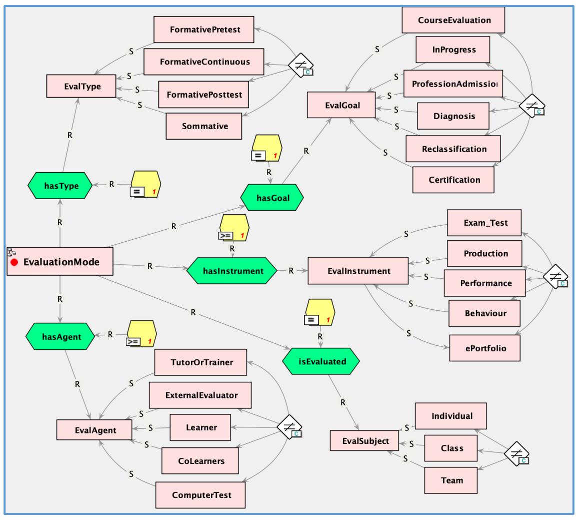


Fig. 4. A GMOT-OWL sub-model for the LD ontology

Each of the main classes in figure 3 has a sign (a little model) in the upper left corner that indicates that it has a sub-model giving more details on the class. One of them is displayed on figure 4) by clicking on the Evaluation Mode class.

This sub-model shows that an evaluation mode has exactly one subject that is evaluated; it can be an individual, a team or a class; these concepts are disjoints. The Evaluation Mode also has one or more agent evaluator, using one or more evaluation instruments. It also has exactly one of six possible distinct goals, and one of four distinct evaluation types: formative pretest, formative post-test, formative continuous or summative evaluation.

The integration between a main model and its sub-models is assured by copying with reference an object (e.g. Evaluation Mode indicated by a red dot). This object is pasted from the main model on figure 4 to the sub-model of figure 5 to which related OWL entities and links can be added. Note that this sub-model can be detailed further on any number of levels, for example adding to *Exam_Test* a sub-model describing a taxonomy of exam types.

The navigation between the set of linked sub-model is facilitated by using the arrows in the left upper corner of the main window or using the Navigator window displaying the tree of sub-models. Other features of this editor allow filtering models to retain only certain types of objects and links, or search for terms that can be distributed in more than one sub-model.

3. UML-based Visual Languages

There is currently a renewed interest in visual languages for ontologies. In this section, we will present the Ontology Definition Metamodel (ODM) and OWLGrED aiming to adapt the graphic Unified Modeling Language [34] to ontology modeling.

3.1. The Ontology Definition Metamodel (ODM)

The Ontology Definition Metamodel (ODM) is a standard specification defined by the Object Management Group [27]. It provides UML metamodels for RDFS, OWL-DL and OWL-Full. Figure 4 displays an example of an ODM model for part of an ontology using some of the complex OWL-DL elements like class intersection and property restrictions.

The readability of this UML 2 diagram is impaired by the fact that rectangles with stereotype annotations in UML classes represent very different OWL elements such as classes, objects properties, individuals or restrictions. To understand the diagram, a human user has to look at the different stereotypes in the boxes and gain an understanding of their meaning. For example, the bottom part of the diagram is meant to express the fact that a "single colored Azalea" is a subclass of Azalea that has exactly one color and a solid color pattern.

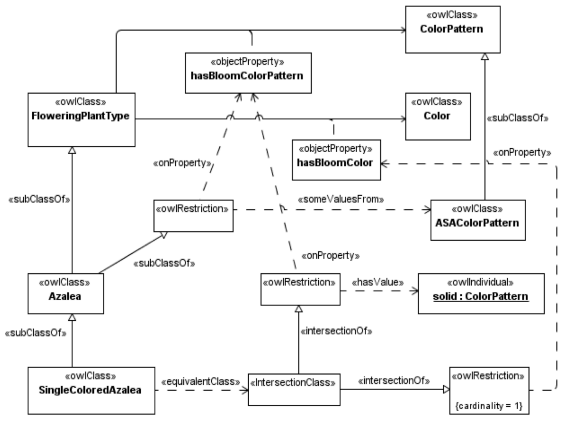


Fig.5. A fragment of an ODM model for an OWL ontology using advanced constructs.[1]

Figure 6 presents a MOT-OWL visual diagram equivalent to the bottom part of figure 6. It uses the same *Exact Cardinality* of 1 and *Has Value* restrictions and reads: "the class of single colored Azaleas is the intersection of two (anonymous) classes, those Azaleas having exactly one color and those Azaleas having exactly one value for color pattern, that of a solid color pattern". Such a visual diagram is more readable because it spares the use of the unnecessary equivalence relation (it will reappear in the OWL-XML translation) and links like "intersection", "on property", "someValue" or "someValueFrom", which are not relations in OWL but elements that serve to construct new classes.

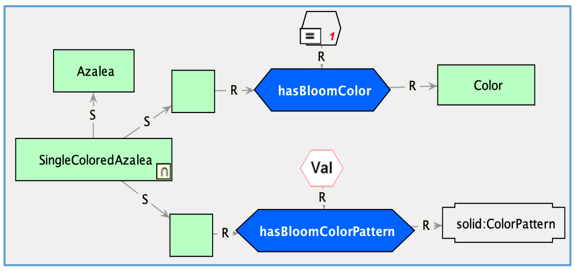


Fig. 6. Fragment of a GMOT model for an ontology.

[1] (OMG, 2014), ODM version 1.1 document, Figure 14.32, p. 182

3.2. OWLGrEd a UML-based editor

OWLGrEd [26] is a tool that provides a UML style graphical notation for OWL 2. Figure 7 presents a fragment of a medicine ontology in this visual notation.

Here object and data properties are visualized as links between UML classes, or as attributes within classes. Data properties are integrated in the class boxes in traditional UML way to reduce the graph complexity but at the expense of *Perceptual Clarity* and *Semantic Transparency*.

This editor qualifies only partially as a visual language because Manchester OWL 2 textual notation replaces visual elements for Data Properties of Object Properties. Furthermore, links between properties cannot be expressed directly.

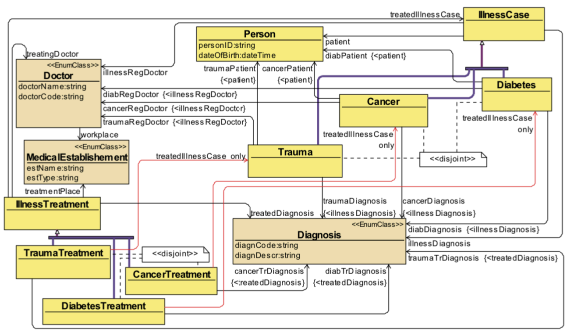


Fig. 7. A views of a OWLGrEd visual model

The MOT-OWL graph on figure 8 offers a totally visual equivalent of the same ontology that is totally visual. It must be completed with sub-models for Diabetes, Cancer and Trauma to provide a complete equivalent to the figure 7 ontology, including the three kinds of corresponding treatments. But the gain here is structural readability an perceptual clarity. The choice to avoid mimicking UML prevents more easily link crossing and, more important, avoiding links that are not ontology relations like “on property”.

3.3. UML as a Visual Ontology Language

According to [7,8], the main reason for using UML notation as a Visual Ontology Language is an effort to integrate ontology development in the mainstream of software engineering, to motivate engineers in using ontologies.

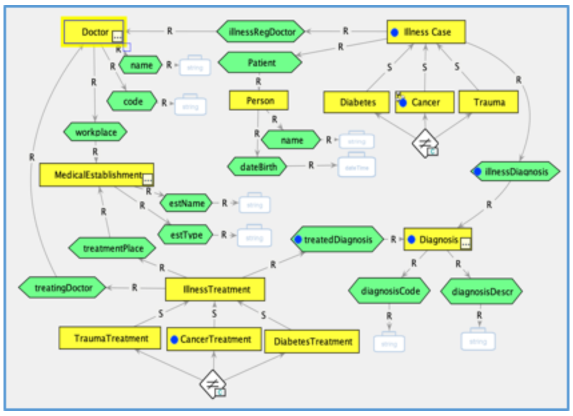


Fig. 8. A MOT-OWL visual model for the medicine example of figure 7.

Unfortunately, UML is based upon an object-oriented paradigm that provides many limitations for ontology visualization. First, the concept of Class in RDFS and OWL is not identical to the concept of Class in UML. Classes in RDFS and OWL are set-theoretic, while object-oriented classes in UML define attributes and methods.

Furthermore, an OWL property represents a relation between subject resources and object resources. It might look similar to the concept of attribute or association in the UML object orientation paradigm, but the owl:ObjectProperty is a standalone concept; it does not depend on any class or resource contrary to associations or attributes in UML. In ontology languages, a property can even be defined with no classes associated with it. That is why a property cannot be represented as an ordinary association or attribute as in object orientated languages.

Gasevic et al. [12] provide an extensive summary of incompatibilities between UML and ontology languages based mainly on [1]. For example, they underline that Ontology languages have the ability to construct classes using Boolean operations (union, intersection and complement) and quantifiers. In UML, there is no corresponding primitives for these notions. Also, in ontology languages one can specify a cardinality constraint for every domain of a property or separately for a range, whereas in UML cardinality must be specified for both association end of a property.

Since our goal is to facilitate the design of ontologies, especially at the initial inception stages, and also their understanding and use at every further stage of the ontology life cycle, the *Semantic Transparency* is key. We believe that the differences between UML and Ontology languages enforce too many unnatural constructions.

4. Non-UML Visual Languages

Ontology engineering tools, such as Protégé , NeOn toolkit or TopBraid Composer offer some visualization functionalities, but do not support a complete or easy-to-use visual ontology modeling capability. Here, we discuss here five non-UML proposals that have been proposed to fill that gap.

4.1. GrOWL visualization tool

GrOWL [18] is an OWL-DL visualization tool implemented as a Java Applet, as a Protégé plugin and a stand-alone Java application. This visual language makes use of the color, shading and shape of nodes to encode properties of the basic OWL constructs (figure 9).

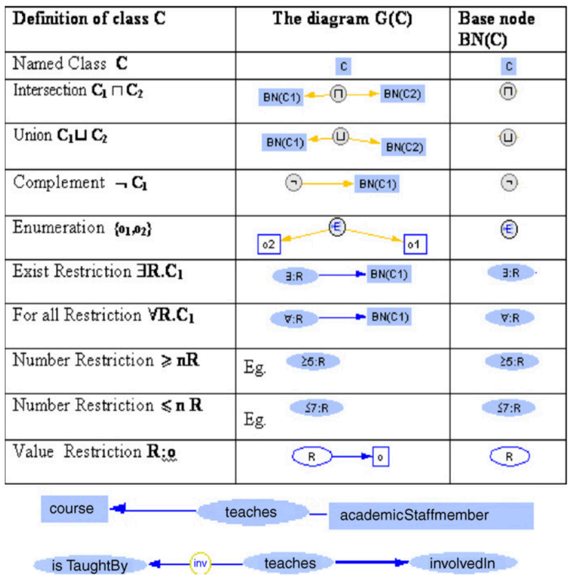


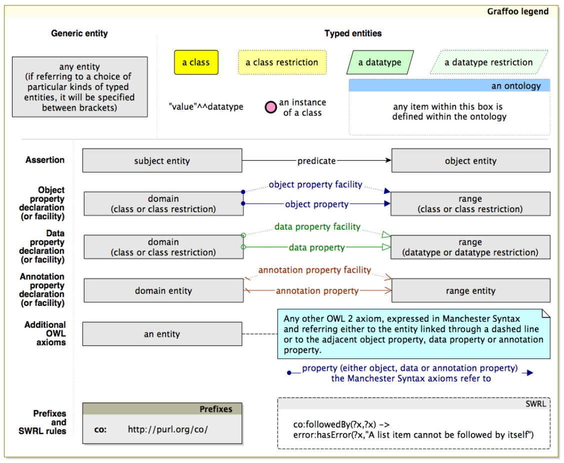
Fig. 9. The GrOWL visual alphabet.

The GrOWL editor is totally visual for OWL 1 ontologies. It does not require to annotate the graphical representation with formulas as with UML editors. It uses some of the DL notations for graph labels which is an advantage for *Semantic Transparency*. But the language did not evolve from OWL 1 to OWL 2. It also uses a large number of symbols that might need to add unnecessary boxes, for example integrating quantifiers in a property precludes using the same property for something else. Also, union or intersection classes should have the same visual symbol as a class.

As mentioned by the authors, the visualization is efficient with small ontologies, but with large ontologies, even the restricted view to a local neighborhood does not guarantee scalability. So, the authors have added filtering mechanism for restricting view to only class definition and its subclasses, superclasses or instances associated to a selected node. They advocate that a combination of filtering and navigation techniques are the most powerful and useful visualization methods.

4.2. The Graffoo Visual Ontology Notation

Graffoo [10] aims at an easy-to-understand notation for OWL, implemented as a GraphML extension for the diagram editor yEd. It does not come with an editing tool, but instead offers a palette for yEd, an open-source editors for graphs. It provides a partial visual notation for OWL ontologies, not based on UML, summarized in figure 10.



Graffoo legend
Generic entity
Typed entities
any entity (if referring to a choice of particular kinds of typed entities, it will be specified between brackets)
a class
a class restriction
a datatype
a datatype restriction
an ontology
"value"^^datatype
an instance of a class
any item within this box is defined within the ontology
Assertion
subject entity
predicate
object entity
Object property declaration (or facility)
domain (class or class restriction)
object property facility
object property
range (class or class restriction)
Data property declaration (or facility)
domain (class or class restriction)
data property facility
data property
range (datatype or datatype restriction)
Annotation property declaration (or facility)
domain entity
annotation property facility
annotation property
range entity
Additional OWL axioms
an entity
Any other OWL 2 axiom, expressed in Manchester Syntax and referring either to the entity linked through a dashed line or to the adjacent object property, data property or annotation property.
property (either object, data or annotation property) the Manchester Syntax axioms refer to
Prefixes and SWRL rules
Prefixes
co: http://purl.org/co/
SWRL
co:followedBy(?x,?x) -> error:hasError(?x,"A list item cannot be followed by itself")

Fig. 10. The Graphoo visual alphabet.

In Graffoo, ontologies are labeled graphs, as in other representations, that use several shapes for nodes and edges to define classes and class restrictions, datatypes and datatype restrictions. The shapes and colors are a bit too similar to promote *Perceptual Clarity*, as we can see on the figure.

Arcs with different colors and shapes surmounted by property names are used to define assertions, annotation properties, data properties, and object properties. The fact that properties are not visual objects by themselves precludes linking them, to express subproperties or inverse properties for example.

Additional axioms in OWL 2 must be added for all those constructs that are not directly supported by a particular graphical element, therefore the notation is not *Totally Visual.* It suffers from some of the same difficulties discussed for UML-based tools that embed textual expressions in the graphical representation.

4.3. VOWL Visual Ontology Editor

VOWL [21,25] is a visual language for the representation of ontologies that aims to be understood by beginning ontology users with only little training. It takes as input a textual OWL ontology created with some other edition tool like Protégé.

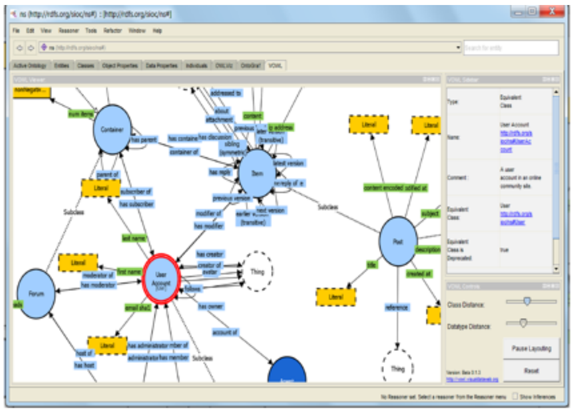


Fig. 11. A user interface of WebVOWL for an ontology.

It is based on a handful of graphical primitives forming the alphabet of the visual language: classes are depicted as circles that are connected by arrows representing the object and datatype properties. Properties and datatypes identifiers are shown in rectangles with different colors. Information on individuals and data values are either displayed in the visualization itself or in a sidebar window as shown on figure 11.The visual elements are combined into a graph that represents central parts of the ontology. They are rendered in a force-directed layout where the size of the circles corresponds to the number of lines around a class. Also, the algorithmic layout of the graph tends to arrange the nodes in a way that the highly connected nodes are placed more at the center of the visualization. The force-directed algorithm can be paused by the user so he can rearrange the graph if he chooses to.

VOWL comes in two versions: a Protégé plug-in and a standalone web application, WebVOWL[2], that provides a number of filters that help reduce the size of the graph in order to focus on certain aspects. This and other features shown on figure 11 aim to support the *Complexity Management* of larger ontologies. Information like disjointness or properties or types of properties like transitivity or symmetry are not displayed visually but listed in the sidebar. Also, equivalent classes are integrated together in the same circle with a double ring for a more readable graph. Inverse properties are displayed to together in boxes on the same double arrow, another contribution to reduce the size of the graph. Despite these functionalities, the visualization of large-scale ontologies will need to be improved as with other visual languages.

VOWL is not a *Totally Visual Modeling* tool. Some editing functionalities are being introduced for basic OWL elements such as owl:allValuesFrom, owl:someValuesFrom, owl:hasValue, but these are not part of the VOWL visualization and must be displayed in a textual way.

4.4. The Graphol Visual Ontology Language and the Eddy Editor

Graphol [5] is a visual ontology language for OWL 2 based on the OWL 2 functional syntax. It allows drawing ontologies in a completely visual way, even to capture complex axioms. Graphol has been proven to be equivalent to OWL 2. Every OWL 2 ontology can be specified in Graphol and conversely, thus satisfying the *Semiotic Clarity principle*, both for

[2] WebVOWL is publicly available at http://vowl.visualdataweb.org

Completeness (1) and *Formality (2)*. The visual ontology editor Eddy [20] enables the design of ontologies with this visual notation.

Graphol is built as an entity-relation graph where some of the nodes (entities) represent basic OWL 2 elements: rectangles denote classes of the ontologies, diamonds denote object properties, circles represent data properties, and rounded rectangles represent data types. Some of the relations for inclusion or equivalence between classes or between properties are represented by solid directed arrows.

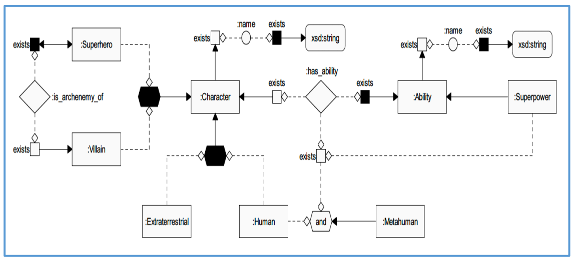


Fig. 12. An OWL 2 ontology expressed in the Graphol visual ontology language

Shown on the example of figure 12 are other kinds of nodes and links that are operators to combine these basic entities using dotted links, in order to build more complex axioms. Black hexagons represent a disjoint union operator grouping two or more predicates. Blank hexagons represent other kinds of operators: or, not, inv and chain. Blank and black square boxes are linked to classes or properties by dotted lines or inclusion links to represent restrictions on the domain (blank squares) or range (black squares) of a property; text like “exists” beside a square box are the type of restriction.

Graphol provides a *Totally Visual* syntax. For example, the left part of the graph on figure 13 displays two existential restriction on the same property :is_archenemy_of, one on the range and the other on the domain. It uses the property, which is itself its inverse property as now permitted by the OWL 2 extension. Superhero is also defined as equivalent to that range and defined as a class for which there exist an archenemy that is a Vilain; and conversely, a Vilain is a class for which there exist an archenemy that is a super hero.

Graphol respects well the *Parsimony principle*. For comparison, we present on figure 14 an equivalent graph in MOT-OWL. True we had to make a referenced copy of the archenemy symmetric property, because MOT-OWL does not still implement the inverse property merger now possible in OWL 2. We have also copied the has_ability property to increase readability. But the MOT-OWL model still uses less nodes than the Graphol model with easier readability.

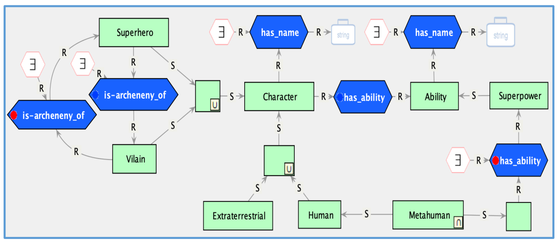


Fig. 13. A MOT-OWL 2 equivalent of figure 12

The authors of Graphol advocate that modeling in their editor has a very short learning curve for DL or OWL experts. But we wonder if mixing basic ontology primitives (classes, properties, individuals, datatypes) with constructing operators respects the *Semantic Transparency* principle. It seems far from the semantic provided by classes as sets and properties as binary relations, notions that are more accessible to most ontology users, especially content experts. For example, a *Metahuman* is understood more directly in figure 14 as the intersection of the sets of all *Human* and the set of persons (anonymous class) that have at least one ability of a *Superpower*.

Also, as the authors point out, the scalability issue will have to be addressed, as with other visual notations, to deal with the *Complex Management* of very large ontologies. Since it is unfeasible to feature thousands of concepts in a single graphical graph, they envisage in their more recent paper to incorporate new functionalities in the editor to support ontology modularization.

4.5. The G-OWL Visual Ontology Language

Our new proposal [15,16] the *Graphical Ontology Web Language (G-OWL)* is based on an Entity-Relation [4] metamodel that serves as a language for the syntax of G-OWL models. This metamodel supports the translation of the G-OWL visual models to other OWL W3C textual syntaxes such as RDF-XML or Turtle. Figure 14 shows the G-OWL metamodel in UML notation with typed entities and relations.

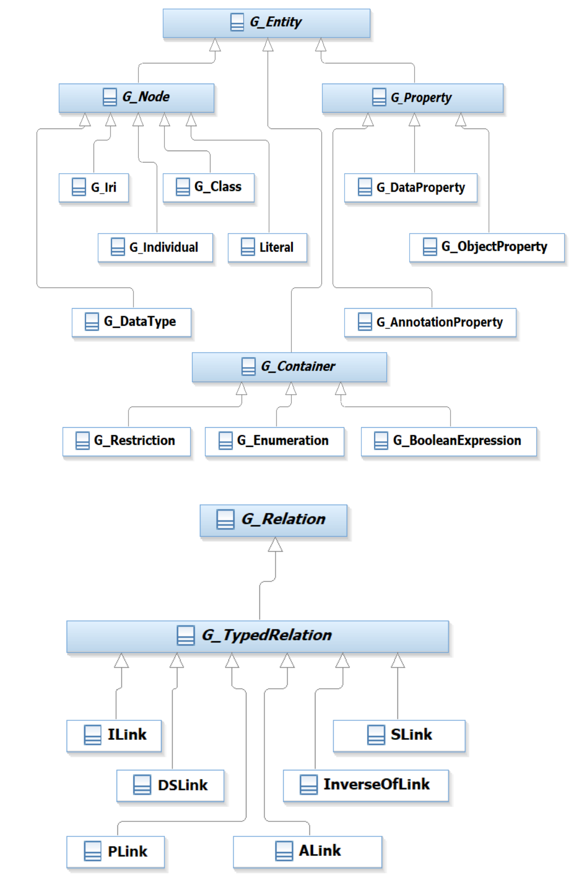


Fig. 14. G-OWL entity-relation metamodel

According to this metamodel, as in MOT-OWL, G-OWL uses polymorphism and a typology of symbols, both to minimize the number of graphical symbols and to enable the interpretation of these symbols that are disambiguated using the topology of neighboring elements.

The *container* visual symbol (a rounded rectangle) reduces the need for many links needed in other visual notations including our previous MOT-OWL. Figure 15 presents three examples of its use. The first container presents an existential restriction, defining a class whose members have at least one value of an "Object property" in the class "Value Class". The second one represents a class which is the union of "n" other classes. The third one asserts the disjointness of "n" data properties.

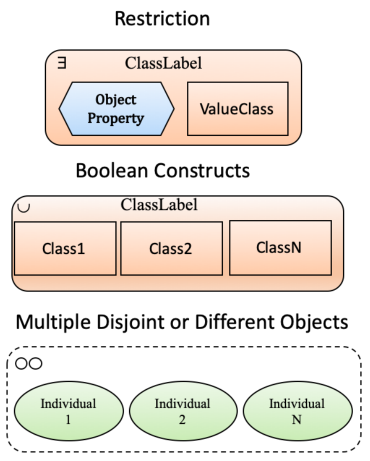


Fig. 15. The use of containers in G-OWL

We beleive that these improvements will increase the *Parsimony* of the notation and facilitate the complexity management of large ontologies.

A complete presentation of the language is out of the scope of this paper, but the following example will give a view of a small ontology. It corresponds to the ontologies for a science-fiction play presented earlier on figure 13 (in Graphol) and figure 14 (in MOT-OWL). The visual model on figure 17 requires fewer links than the other two models thanks to the use of containers for restrictions and Boolean operations.

First, we define *Vilains* as persons who have some *Super-Hero* as their *archenemy*. Then, we add the precision that a *Super-Hero* is a person who has a *Vilain* as an *archenemy*. Both of these classes are subclasses of *Characters*. So are the classes of *Extra-terrestrials* and *Humans*.

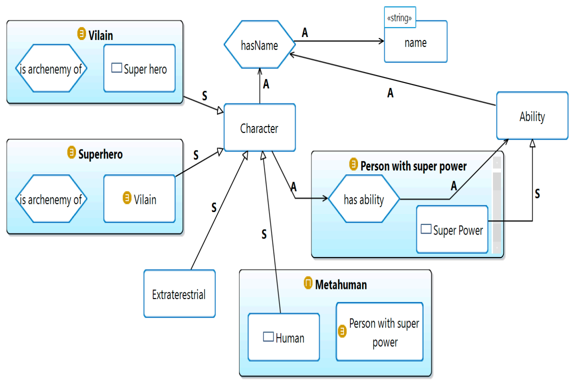


Fig. 16. An equivalent G-OWL model to figure 13, 14.

Characters have two properties, they have a *name*, which is a string given by a data property, and one or more abilities specified by the *has_ability* object property. *Abilities* also have a name. Note that **A** links (similarly to **R** links in MOT-OWL) serve to relate a property from its domain and to its range.

Finally, for the notion of a *Metahuman*, we first had to define the notion of a person with super powers, by using another OWL existential restriction. *Super powers* are a subclass of a character's abilities. Then the class of Metahumans is defined precisely as the intersection of the class of Humans and of the class of persons with at least one super power.

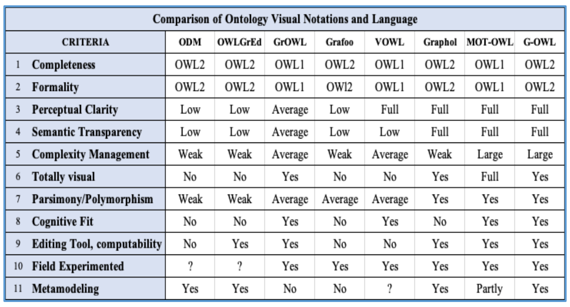
5. Comparison and Analysis

The following table compares the general features of the visual notations discussed so far. To the nine criteria derived from Moody's work on the Physics of Notation Theory (PoNT) presented in the introduction, we add two more criteria:

- **Field experimented.** According to the documentation, a field experimentation has been conducted with various levels of users.
- **Metamodeling.** A metamodel of the notation exists, so that the notation is independent from its software implementations.

The information in table 1 must not be interpreted more than what it is: a brief evaluation of the visual notations based on the analysis of one or two documents for each notation. Furthermore, some of these notations may have evolved since the publication of the papers we have consulted.

Table 1 – Comparisons of the Visual Notations and Languages.



Comparison of Ontology Visual Notations and Language

	CRITERIA	ODM	OWLGrEd	GrOWL	Grafoo	VOWL	Graphol	MOT-OWL	G-OWL
1	**Completeness**	OWL2	OWL2	OWL1	OWL2	OWL1	OWL2	OWL1	OWL2
2	**Formality**	OWL2	OWL2	OWL1	OWl2	OWL1	OWL2	OWL1	OWL2
3	**Perceptual Clarity**	Low	Low	Average	Low	Full	Full	Full	Full
4	**Semantic Transparency**	Low	Low	Average	Low	Low	Full	Full	Full
5	**Complexity Management**	Weak	Weak	Average	Weak	Average	Weak	Large	Large
6	**Totally visual**	No	No	Yes	No	No	Yes	Full	Yes
7	**Parsimony/Polymorphism**	Weak	Weak	Average	Average	Average	Yes	Yes	Yes
8	**Cognitive Fit**	No	No	Yes	No	Yes	No	Yes	Yes
9	**Editing Tool, computability**	No	Yes	Yes	No	No	Yes	Yes	Yes
10	**Field Experimented**	?	?	Yes	Yes	Yes	Yes	Yes	Yes
11	**Metamodeling**	Yes	Yes	No	No	?	Yes	Partly	Yes

Our goal in this section is not to find the best notation (if it ever exists) but to identify possible improvements for our own Ontology Visual Languages and Editors. This comparison helps reveal the actual strengths and weaknesses of the MOT-OWL notation and identify guidelines for improvements to our new G-OWL Visual Ontology Language. To achieve this, we will now discuss each group of criteria.

5.1. Completeness and Formality

Both MOT-OWL (for OWL-DL) and G-OWL (for OWL 2) obey the first two principles of *Completeness (1)* and *Formality (2).* To each semantic OWL object correspond a unique symbol or an understandable combination of symbols in the visual language. Conversely, to each visual symbol (or some combination of symbols) correspond only one semantic OWL 2 object that can be disambiguated using the symbol's visual context. For example, even if the same **S** link used between classes or properties (polysemy) it will correspond to a different OWL relation: a subclass relation when used between two classes, or to a sub property relation when used between two properties.

However MOT-OWL respects completeness only with regards to OWL-DL since it has not been be extended to most of the new OWL 2 features [37] For example, the OWL 2 *DisjointUnion* would require a new class operation symbol to prevent using every time two constructs, the union n-ary construct and the multiple pair-wise disjoint symbol implemented in the MOT-OWL notation.

Most of the new OWL 2 features have been implemented in the G-OWL syntax and its Onto-Case4G-OWL Eclipse-based editor. An important point to underline here is the decision not to implement a strict one-t-one correspondence between OWL 2 semantic components and the MOT-OWL or G-OWL set of symbols. This provides a more readable representation and an easier modeling of ontologies.

5.2. Perceptual Clarity and Semantic Transparency

Some positive aspects of both MOT-OWL and G-OWL is that they are *Totally Visual(6),* have been *Field Experimented (10),* and have a good Cognitive Fit (8) with content experts and ontology modelers. As with most of the other non-UML notations, they provide a high level of *Perceptual Clarity (3)* since, all the OWL basic entities are represented by symbols with distinct shapes and colors. Looking at any ontology visual model, it is possible to rapidly recognize the basic entities and their relationship.

Semantic Transparency (4) in MOT-OWL and G-OWL are favored by the use of mathematical symbols for the existential and universal restrictions and for the Boolean constructed. The direction of the "ruled by" **R** link (in MOT-OWL) and the **A** link (in G-OWL) from a class to a property, or from a property to a class, individual or value, suggests better the semantics of a property as a relation between two classes, the first being its domain and the second being its range. A similar semantic suggestion is in the **S** subclass links from two or more classes to their union class (in MOT-OWL).

A weakness in MOT-OWL has been corrected in G-OWL. Data Properties and Object Properties must be represented with lightly different shapes and color, "lightly" because they are both relations (or predicates) in the set theoretic view of FOL and Description logics. As it is now, if a property is declared without a range, the editor will not be able to decide if it is an object property or a data property.

Another problem is in the class complement represented by a bi-directional link in MOT-OWL. Since the complement of class is another class, G-OWL aim to represent this notion using a container (not implemented yet). The container is itself a class that is the complement of "this class", a visual objects that can be linke to order visual objects.

5.3. Complexity Management and Polymorphism

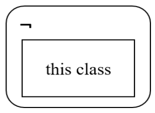
We will now discuss two important related requirements: *Complexity Management (5)* and *Parsimony/Polymorphism (7).* As mentioned before there is in general an important scalability issue with visual models. There are three directions to solve the scalability problem: modification of language itself, simplifications in the user interface of the editing tools and the decomposition of the global model into interrelated sub-models.

The preoccupation for simplification of the language is shown in some of the new features integrated in OWL 2 for multiple *DisjointClasses* that state that all classes are pairwise disjoint, instead of declaring disjointness for each pair in a large collection. Other new features such as *Disjoint Propert*ies or *Property-ChainInclusion, Top and Bottom Properties*, *Property Qualified Cardinality Restrictions* or the use *ObjectInverseOf* will reduce the number of links between ontology objects.

Reduction in the size of models can also be implemented in a parsimonious user interface making full use of polymorphism. In addition, enclosing identical individuals, equivalent classes or properties, Boolean constructs or inverse properties in more visually distinct G-OWL containers will eliminate the need of many links in the model presentation, while permitting the correct semantic translation in one of the standard textual OWL syntax.

Finally, probably the best way to implement the *Complexity Management (5)* principle is to facilitate model decomposition into linked sub-model. One example is given at the end of section 2. It is a solution we have experienced in some large MOT modeling projects such as building an Instructional Engineering Method (MISA) or modeling the architecture for a model-driven development of the TELOS platform [31].

Other features of the GMOT-OWL or G-OWL editors facilitate also *Complexity Management (5)* by filtering the model by object or link types or by searching for terms that can be repeated in various sub-models throughout the overall ontology.



Following the example of GrOWL, the filtering mechanisms should be improved by providing local displays, for example of a class with its sub-classes, super-classes and linked properties. This is now possible in the G-OWL editor.

G-OWL has been implemented on top the Eclipse IDE and provides many interesting features for Complexity Management, including the possibility to organize large ontology visual models into an integrated set of significant views. These possibilities will need to be explored further.

5.4. Metamodeling Issues in MOT-OWL and G-OWL

Metamodeling in a visual ontology language plays a central role in Model-Driven software engineering [28]. The principle is to use design models of a software application and apply code generators to the model so that the software functionalities are platform independent. A metamodel is an explicit description - a language with a precise vocabulary and grammatical rules – that specifies how each compliant model can be built. A metamodel must provide a formalized specification of the visual notation for OWL 2 ontologies.

MOT-OWL has a partial metamodel using the MOT language itself [30] to describe the entities and relations in its vocabulary, and a set of rules to combine them, describing the grammar of the visual language. But this metamodel is not related to the mainstream of Model-driven architectures that proposes the use of a meta-metamodel, a language to write metamodels. In this way a formal metamodel can be specify for the language in order to promote its interoperability with other notations (visual or textual) for ontologies or compare formally its span of applications.

This work has been realized for the development of the G-OWL visual language and its OntoCase editor by using the Eclipse Modeling Framework (EMF) as the formal metamodel modeler.

6. Conclusion

Throughout this presentation, we have used the Physics of Notations Theory to evaluate a variety of visual languages for ontologies that have been proposed in the literature, including our own MOT-OWL and G-OWL. We have adapted the nine principles in Moody's work to state eleven principles for visual ontology languages.

Using these principles, we have first examined the UML-based proposals. Since our main goal is to facilitate the design of ontologies, especially at the initial inception stages, and also their understanding and use at every further stage of the ontology life cycle, we have identified important differences between UML object-oriented paradigm and Ontology languages that enforce too many unnatural constructions. The semantic transparency of an ontology visual notation is crucial and UML forces the use of too many textual elements either within classes or on links between classes, adding a multiplicity of links that render difficult the understanding of even small models and the management of large models.

We have presented five other visual notation that propose a number of interesting innovations for a visual language, particularly the VOWL and Graphol proposals that seem most interesting, though quite different. We have identified some of the ideas that would help improve our own visual languages. Many new ideas have been implemented in an editor for the new G-OWL visual language that has been partly field tested.

References

[1] Baclawski, K., Kokar, M., Kogut, P., Hart, L., Smith, J.E., Letkowski, J., & Emery, P. (2002). "Extending the Unified Modeling Language for ontology development," Software and Systems Modeling, vol. 1, no. 2, pp. 142–156.

[2] Berners-Lee, T., Hendler, J., Lassila, O. (2001) The semantic web. Scientific American 284(5), 34

[3] Brockmans, S., Volz, R., Eberhart, A., & Löffler, P. (2004). "Visual modeling of OWL DL ontologies using UML," Proceedings of the 3rd International Semantic Web Conference, Hiroshima, pp. 108–213.

[4] Chen, P. (2002) Entity-Relationship Modeling: Historical Events, Future Trends, and Lessons Learned, in *Software Pioneers: Contributions to Software Engineering*, ed: Springer, 2002, pp. 297-310.

[5] Console, M., Lembo L., Santarelli V. and Savo D.F. (2017). Graphol : Ontology Representation Through Diagrams. http://www.obdasystems.com/sites/default/files/2017-03/DL-2013_3.pdf , consulted March 15, 2020.

[6] Corcho, O., M. Fernández-López, and A. Gómez-Pérez, Ontological Engineering: Principles, Methods, Tools and Languages, in *Ontologies for Software Engineering and Software Technology*. 2006. p. 1-48.

[7] Cranefield, S. (2001a). "Networked knowledge representation and exchange using UML and RDF," *Journal of Digital Information*, vol. 1, no. 8, article no. 44, 2001-02-15.

[8] Cranefield, S. (2001b). "UML and the Semantic Web," *Proceedings of the Semantic Web Working Symposium*, Stanford University, CA, pp. 113–130.

[9] Denny, M. (2002), Ontology Building: A Survey of Editing Tools. http://www.xml.com/pub/a/2002/11/06/ontologies.html

[10] Falco, R., Gangemi, A., Peroni, S., Shotton, D. and Vitali, F. Modelling OWL Ontologies with Graffoo, in *The Semantic Web: ESWC 2014 Satellite Events*. vol. 8798, V. Presutti, E. Blomqvist, R. Troncy, H. Sack, I. Papadakis, and A. Tordai, Eds., ed: Springer International Publishing, 2014, pp. 320-325.

[11] Fernández-López, M., Gómez-Pérez, A., Sierra, J.P., & Sierra, A.P. (1999), Building a chemical ontology using Methontology and the Ontology Design Environment, *IEEE Intelligent Systems*, vol. 14, no. 1, pp. 37–46.

[12] Gašević, D., Djurić D. and Devedžić, V. (2006) Model Driven Architecture and Ontology Development. Springer-Verlag, New York.

[13] Genon, N., Heymans, P. and Amyot, D. (2011) Analysing the Cognitive Effectiveness of the BPMN 2.0 Visual Notation. In: Malloy, B., Staab, S., van den Brand, M. (eds.) SLE 2010. LNCS, vol. 6563, pp. 377–396. Springer, Heidelberg

[14] Haase, P., Lewen, H., Studer, R. Tran, D. T., Erdmann and d'Aquin M. (2008) "The Neon ontology engineering toolkit," http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.140.7081&rep=rep1&type=pdf

[15] Héon, M. (2014) Web sémantique et modélisation ontologique (avec G-OWL). Edition ENI (France). 444 pages.

[16] Héon, M., Nkambou, R. and Langheit, C. (2016) Toward G-OWL: A Graphical, Polymorphic And Typed Syntax For Building Formal OWL 2 Ontologies, presented at the Proceedings of the 25th International Conference Companion on World Wide Web, Montréal, Québec, Canada, 2016. 3.

[17] Kendall, E.F. and D.L. McGuinness, Ontology engineering. Synthesis Lectures on The Semantic Web: Theory and Technology, 2019. 9(1): p. i-102.

[18] Krivov, S., Williams, R. and Villa, F. (2007). GrOWL: A tool for visualization and editing of OWL ontologies. *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, pp. 54-57.

[19] Larkin J. H. and Simon H. A., (1987) Why a diagram is (sometimes) worth ten thousand words. *Cognitive science*, vol. 11, pp. 65-100, 1987.

[20] Lembo, D., Pantaleone, D. , Santarelli, V. & Savo, D. F. (2018). Drawing OWL 2 ontologies with Eddy the editor. AI Communications. 31. 1-17. 10.3233/AIC-180751.

[21] Lohmann, S., Negru, S., Haag, F. and Ertl, T. "Visualizing ontologies with VOWL", *Semantic Web*, vol. 7, pp. 399-419, 2016.

[22] Mizoguchi, R. & Kitamura, Y. (2001). "Knowledge systematization through ontology engineering – a key technology for successful intelligent systems," Invited paper, Pacific-Asian Conference on Intelligent Systems, Seoul

[23] Moody, D. (2009) The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering, *IEEE Transactions on Software Engineering*, vol. 35, pp. 756-779.

[24] Moody, D. L. and van Hillegersberg, J. (2008) Evaluating the Visual Syntax of UML: An analysis of the Cognitive Effectiveness of the UML Family Of Diagrams. Conference on Software Language Engineering.

[25] Negru, S., Lohmann, S. and Haag, F. *VOWL Specification of Version 2.0*, 7 April 2014, http://purl.org/vowl/spec/v2/ retrieved March 15,2020

[26] Ovcinnikova, J. and Cerans, K. (2016). "Advanced UML Style Visualization of OWL Ontologies." http://www.divaportal.org/smash/get/diva2:1033953/FULLTEXT02#page=144, retrieved March 15, 2020

[27] OMG (2014). Ontology Definition Metamodel (ODM): OMG Adopted Specification, version 1.1. Available: http://www.omg.org/spec/ODM/1.0/Beta2/PDF/ retreived March 15, 2020.

[28] OMG (2003) *MDA Guide Version 1.0.1*, OMG Document Number: omg/2003-06-01", OMG, 12.6.2003, http://www.omg.org/cgi-bin/doc?omg/2003-06-01

[29] Paquette G. (2008). Graphical Ontology Modeling Language for Learning Environments. *Technology, Instruction., Cognition and Learning* , Vol.5 , p.133-168, Old City Publishing, Inc.

[30] Paquette, G. (2010) Visual Knowledge and Competency Modeling - From Informal Learning Models to Semantic Web Ontologies. Hershey, PA: IGI Global, 2010.

[31] Paquette, G., Rosca, I., Mihaila, S. and Masmoudi, A. (2007) "TELOS: A service-oriented framework to support learning and knowledge management," *in E-Learning Networked Environments and Architectures*, Springer, 2007

[32] Popescu, G. and Wegmann A. (2014) Using the Physics of Notations Theory to Evaluate the Visual Notation of SEAM. CBI (2) 2014: 166-173. https://infoscience.epfl.ch/record/198951/files/PID3224947.pdf, consulted March 15, 2020.

[33] Protégé (2009). Welcome to Protege. http://protege.stanford.edu/

[34] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. E. (1991) Object-oriented modeling and design vol. 199: Prentice-hall Englewood Cliffs, NJ.

[35] Staab, S. & Studer, R., eds. (2004) *Handbook on Ontologies*. Springer, Berlin, Heidelberg.

[36] TopBraid (2017). TopBraid Composer (TM). Available: https://www.topquadrant.com/tools/modeling-topbraid-composer-standard-edition/

[37] W3C (2012) OWL 2 Web Ontology Language, New Features and Rationale (2nd Edition), 11 December 2012, http://www.w3.org/TR/2012/REC-owl2-new-features-20121211/, consulted March 15, 2020.