G-OWL: A Complete Visual Syntax for OWL 2 Ontology Modeling and Communication

Michel Héon [a] [0000-0001-7515-6382] and Gilbert Paquette [b] [0000-0002-2898-3462]

[a] *Cotechnoe Inc and UQAM Université du Québec à Montréal, Montréal, Canada, heon@cotechnoe.com*
[b] *LICEF Research Institute, Université TELUQ, Montréal, Canada, gilbert.paquette@licef.ca*

Abstract. Semantic web ontologies are usually modeled using standard text-based syntaxes such as OWL/XML, Functional, Manchester or Turtle. Over recent years, there has been an increasing need for representing ontologies visually to help ontological engineers or modelers represent elicited knowledge from domain expert, big data, model data structures or simply present data schemas and metadata to general users. We believe a visual representation is an essential way for understanding knowledge and to help elaborate formal ontologies for their communication and their use by humans. In this paper, we present the Graphical Ontology Web Language (G-OWL), a visual syntax for the graphical modeling and visualization of OWL 2 or RDFS ontologies. In line with previous research in cognitive science, G-OWL uses syntactic and semantic principles that simplify both its use and its interpretation. Indeed, the use of typology and polymorphism makes it possible to minimize the number of visual signs in a grammar, thus reducing the cognitive load on users, while preserving the formal character of the ontology. This G-OWL visual syntax is integrated in a software tool called OntoCASE4G-OWL to support the elaboration of ontologies and their translation to standard text-based syntaxes such as Turtle. This paper aims to present the definition of the G-OWL visual syntax and to demonstrate its highly readable character through an objective assessment of criteria such as: semiotic clarity, semantic transparency and graphic complexity. The G-OWL visual syntax will also be compared with other visual syntaxes and will be evaluated in order to measure its highly human-readability in reading activities, modeling or new knowledge deduction by novice, intermediate and expert in ontology modeling.

Keywords: Semantic web, ontology modleing, ontology engineering, visual ontology, visual ontology modeling theory, visual knowledge representation, visual notation evaluation, web ontology language

1. Intrroduction

The Semantic Web has witnessed huge developments since its inception at the beginning of the century. Ontologies are now in use in many fields such as medicine, science, e-commerce, as well as in library and educational applications, to name but a few, and it addresses both organizational needs [1] for knowledge representation and knowledge computation. The rapid growth of the Web of Linked Data, and the necessity of knowledge representation requires new skills and tools from users to visualize the structure of an ontology and for modelers to assure its inception and evolution.

In this paper, we present a human-readable visual concrete syntax, the *Graphical Web Ontology Language* (G-OWL), which has the following characteristics: 1) its syntax is completely visual; 2) its semantic is easily interpretable by humans from the visual representation; 3) its symbols have semantic correspondents in W3C recommended semantic web syntax and the visual graph can be exported to W3C standard text-based machine-readable representations, while being more readily interpretable; 4) compared to semantic web ontology language its syntax contains a limited number of visual symbols to be easily manageable for modeling and communication to human readers and designers.

1.1. *The OWL 2 Standard Textual Syntaxes*

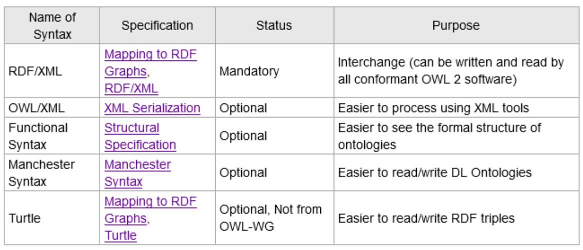
Recommended by the World Wide Web Consortium (W3C), the Web Ontology Language

(OWL) [2, 3] has now become the “lingua franca” for the definition of ontologies and for software engineering based on ontologies. According to section 2 of the OWL 2 Document Overview [2], an ontology contains two layers: the semantic layer that retains its meaning and the syntactic layer that captures the concrete notation enabling its serialization.

From the beginning, despite its RDF [4] and RDFS [5] graph basis, the main preoccupation of the W3C has focused on enabling machine readability and securing its use in software applications. Thus far, the formal concrete syntax for OWL [2, 3, 6] recommended by the W3C, such as OWL/XML [7], Turtle [8], Functional syntax [9], or Manchester Syntax [10] are all text-based syntaxes readable only by skilled computer scientists (Table 1).

Table 1

Purpose of W3C Semantic Web Concrete Syntax Specifications



Name of Syntax	Specification	Status	Purpose
RDF/XML	Mapping to RDF Graphs, RDF/XML	Mandatory	Interchange (can be written and read by all conformant OWL 2 software)
OWL/XML	XML Serialization	Optional	Easier to process using XML tools
Functional Syntax	Structural Specification	Optional	Easier to see the formal structure of ontologies
Manchester Syntax	Manchester Syntax	Optional	Easier to read/write DL Ontologies
Turtle	Mapping to RDF Graphs, Turtle	Optional, Not from OWL-WG	Easier to read/write RDF triples

The linear textual descriptions involved in these standards blur the structure of the ontology and makes it difficult to design new ontologies. Alternatively, ontologies are sometimes represented using limited graphs for explanation purposes, but there is yet no W3C recommendation for a formal visual concrete syntax for OWL 2 ontology edition, modeling or visualization.

1.2. *Ontology Visual Syntaxes*

It is usually agreed upon that a visual notation is more easily readable and understandable than a text-based representation. In the case of formal representations [11] that can be processed by computers such as Unified Modeling Language (UML) [12] or the Business Process Modeling Notation (BPMN) [13], their interpretation seems more difficult for novice users, as shown by some experiments [14, 15]. Contrary to semi-formal visual languages, formal representations require that each visual symbol has a unique non ambiguous meaning that must be respected.

For these reasons, there is currently a renewed interest in visual notations for ontologies. Actually, popular ontology engineering tools, such as Protégé [16], NeOn toolkit [17] or TopBraid Composer [18] offer many visualization functionalities, but do not support a complete or easy-to-use visual ontology modeling that would facilitate ontology design, development and use. Also, various proposals have been made such as ODM [19, 20], VOWL [21], OWLGrEd [22, 23], GrOWL [24], Graffoo [25], Eddy/Graphol [26, 27], and our own MOT-OWL[28, 29]. To date, no visual notation has clearly emerged or has achieved wide usage.

In his study of visual notations such as UML and BPMN, Moody [30] states some design principles for a Physics of Notation Theory (PoNT) that are particularly useful for the present research. In a previous study[1], we have applied these principles to evaluate and compare the various proposals for an Ontology Visual Language. This preliminary study has helped us uncover guidelines for the G-OWL language to be presented here.

1.3. *A Solution Inspired by Semi-Formal Notations*

Some field studies show that design and interpretation efficiency of visual notations is increased when the visual notation is more flexible, such as in semi-formal visual notations [11], such as Mind Mapping [31], Concept Mapping [32] or Modeling using Object Types (MOT) [33, 34]. These notations have proven their cognitive efficiency [35-37] and a high level of human readability for Visual Knowledge Modeling. They can be used at the early stages of ontology design to promote knowledge transfer [38] from content experts to computer

[1] This study entitled *Designing and Communicating Ontologies Visually* is being evaluated for publication in the IOS Semantic Web Journal.

scientist in organizations or to support learning design [35] for education and training. Afterwards, the semiformal graphs provide a basis for the more formal design that is needed for an ontology.

Some common properties of semi-formal visual notation (e.g. polysemy) might explain their efficiency. They provide a minimal number of visual symbols. According to Miller [39] a limited number of symbols decreases the cognitive load [40] involved in their processing. As a consequence, polysemic semi-formal notations increase the semantic content of each symbol in the mind of their users, helping them accept some syntactic ambiguity that they can disambiguate.

For example, in the MOT language, there is a total of 8 visual symbols for entities and 6 kinds of links between them. Various geometric forms represent types of knowledge such as facts, concepts, procedures or principles and their instances. Oriented arrows represent various types of links between them. The same symbol for specialization links is used between concepts, procedures or principles. The different meanings involved are disambiguated by the context. Also, the input/product link between a concept and a procedure is interpreted differently according to its direction, as "input" from concept to procedure, or as "product" from procedure to concept.

Our goal in this paper is to apply similar principles to represent visually the OWL 2 languages.

1.4. *Previous Work on Visual Language and Tools*

Our multidisciplinary research in the field of visual modeling started at the end of the nineties [41], leading to a set of visual knowledge modeling languages and editing tools: MOT [34, 41, 42], MOTplus [43] and G-MOT [44, 45] based on typed knowledge entities and links. During the earliest phase of our research program, the goal was to synthesize various visual formalisms such as conceptual maps, flowcharts, or decision trees into a unified visual language that could be accessible to non-computer scientists such as educators or managers in organizations. To achieve this task, we based our research on the visual systems created for the analysis and design of information systems, such as Chen's Entity-Relationship model [46], Sowa's Conceptual Graphs [47], the Object Modeling Technique (OMT) [12, 48], KADS [49] and the UML [12].

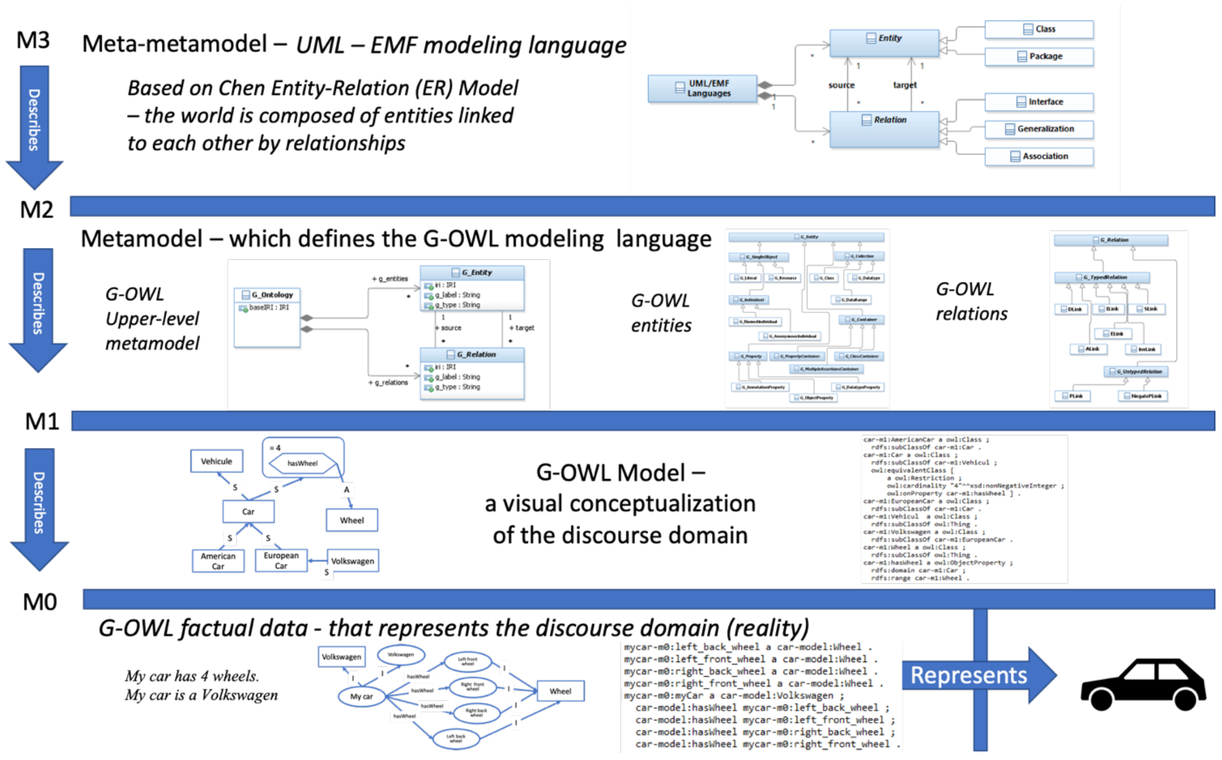
In our search for a user-friendly visual syntax, we uncovered a consensus for knowledge categories in educational science [50, 51], despite slightly different terminologies, based on four basic types of knowledge entities: facts, concepts, procedures, and principles. All four types of knowledge were also clearly identified within the framework of schema theory [52], which plays a central role in Artificial Intelligence (AI) and Cognitive Science in general. This typing of knowledge entities is the basis for our MOT visual representation syntax [53].

With these primitives in the MOT visual syntax, we have been able to build, in various projects and fields, complex models such as conceptual maps, decision trees, workflows, methods, and theories. An instructional design methods such as MISA [54], was entirely described graphically using the MOT modeler, leading to the construction of an instructional design workbench for learning environments.

Another interesting category of models built with MOT modelers are "laws and theories," of which a particular case is an ontology. So, when the W3C published the first OWL-DL [55] document in 2004, we were ready to start specialize the MOT language as a Visual Ontology Language: MOTplus/OWL editing tool in 2008, and, the GMOT/OWL modeler, in 2012. The later served to build an executable model of TELOS a semantic web based system [56].

The MOT visual syntax and the G-MOT/OWL modeler are the direct ancestors of the G-OWL [57-59] visual syntax and of OntoCASE4G-OWL modeling tool [60] presented here. G-OWL is thus deeply rooted in cognitive science, artificial intelligence, and software engineering research.

In section 2 and 3, we will present the metamodel and the visual syntax and semantics of the Graphical-Ontology Web Language G-OWL. Section 4 will summarize the visual theory principles that will serve to evaluate the G-OWL language proposal in section 5, comparing it with two other visual notation proposals. Section 6 will extend this discussion further and prepare the conclusions in section 7.



```
car-m1:AmericanCar a owl:Class ;
  rdfs:subClassOf car-m1:Car .
car-m1:Car a owl:Class ;
  rdfs:subClassOf car-m1:Vehicul ;
  owl:equivalentClass [
      a owl:Restriction ;
      owl:cardinality "4"^^xsd:nonNegativeInteger ;
      owl:onProperty car-m1:hasWheel ] .
car-m1:EuropeanCar a owl:Class ;
  rdfs:subClassOf car-m1:Car .
car-m1:Vehicul  a owl:Class ;
  rdfs:subClassOf owl:Thing .
car-m1:Volkswagen a owl:Class ;
  rdfs:subClassOf car-m1:EuropeanCar .
car-m1:Wheel a owl:Class ;
  rdfs:subClassOf owl:Thing .
car-m1:hasWheel a owl:ObjectProperty ;
  rdfs:domain car-m1:Car ;
  rdfs:range car-m1:Wheel .
```

```
mycar-m0:left_back_wheel a car-model:Wheel .
mycar-m0:left_front_wheel a car-model:Wheel .
mycar-m0:right_back_wheel a car-model:Wheel .
mycar-m0:right_front_wheel a car-model:Wheel .
mycar-m0:myCar a car-model:Volkswagen ;
  car-model:hasWheel mycar-m0:left_back_wheel ;
  car-model:hasWheel mycar-m0:left_front_wheel ;
  car-model:hasWheel mycar-m0:right_back_wheel ;
  car-model:hasWheel mycar-m0:right_front_wheel .
```

Fig. 1: G-OWL Metamodel in the MDA Framework

2. Metamodeling the G-OWL Language

We now presents the process followed for the definition of the G-OWL visual notation for the OWL 2 ontology web language. Fig. 1 (above) positions the G-OWL language in the MDA Modeling Development Architecture framework [61].

2.1. *The G-OWL position in the MDA framework*

At the top meta-metamodel level (M3), we have used the Eclipse Modeling Framework (EMF) [62], an UML-like modeling language to define G-OWL's metamodel. This language is based on Chen's entity-relation meta-metamodel that envision the world as composed of entities linked together by relations. This meta-metamodel is materialized by the creation of the g-owl.ecore file [62].

As shown in the M3 part of Fig. 1, a UML-EMF language is composed of a set of entities and relations. Each relation has exactly one entity as its source and one entity as its target. An entity can be the source of many relations and also the target of many relations.

Interface, generalization and association are sort of UML-EMF relations between entities. Class and Package are examples of entities that compose a UML-EMF language.

At the metamodel level (M2), EMF has been used, to define the G-OWL modeling language using UML-EMF constructs. Since UML-EMF is the language of the Eclipse Development System, a visual modeler for the language entitled OntoCASE4G-OWL [60] has been developed on this basis using Eclipse and other tools. The UML-EMF graphs shown on the second level of Fig. 1 (M2) will be explained in the following sub-sections.

At the model level (M1), G-OWL ontologies are created as visual models using the G-OWL language defined in M2 to express a visual conceptualization of a discourse domain. An example of G-OWL visual model is shown on this part of Fig. 1, together with a corresponding serialization in the Turtle OWL 2 textual notation. Thanks to the G-OWL metamodel, serialization and deserialization to and from Turtle OWL concrete syntax can be performed.

The factual data model level (M0) group assertions about the world that can also be presented using the G-OWL language that covers both OWL 2 conceptualization and assertions in RDF. A small RDF graph in G-OWL visual syntax is presented at this M0 level, together with its equivalent in the Turtle notation. At this level, there a two-representation possible. If the M1 ontology is imported, the OntoCASE4G-OWL modeler can associate an RDF resource to any individual, class or property present in the ontology. If the ontology is not imported, the modeler considers the resources and the predicate as a simple IRI to serve in an RDF triple.

2.2. *G-OWL High Level Metamodel*

We now present the components of the M1 metamodel for the G-OWL language and some generic visual symbols of the visual language.

The *G-OWL Metamodel* contains *Constructors* which serve to represent the visual notation semantics into G-OWL visual syntax.

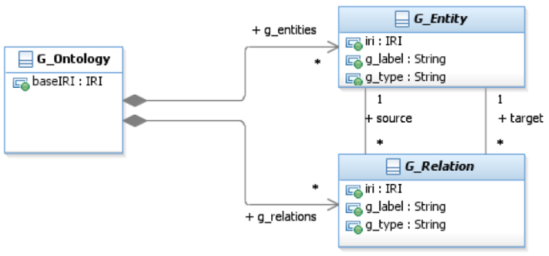


Fig. 2. G-OWL Meta-metamodel

As an UML-EMF type of language, the G-OWL notation is composed of two abstract classes: `G_Entity` and `G_Relation` (**Erreur ! Source du renvoi introuvable.**. Each member of these classes has an IRI address on the Web, a type and a lable.

Each `G_Relation` has a `G_Entity` source and a `g_entity` target. From this simple abstract definition, it is possible to define the set of basic visual symbol of G-OWL).

Part a) of Fig. 3 shows that a visual entity symbol (`G_Entity`) has two attributes: `g_type` and `g_label` that will be included in the visual form representing the entity (not necessarily a rectangle). The `g_type` indicates the type of objects that is represented; for example, a “F” in the polygon of an ObjectProperty indicates the type *functional* for that property, while a “ ∀ ” in a container represents a universal restriction property.

A `g_label` in a visual form assigns to the entity a term chosen by the user, that will be serialized as a `rdfs:label`.

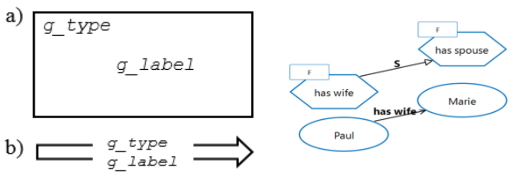


Fig. 3. Label and Type attributes for G-OWL Symbols

Part b) of Fig.3 presents the case of a `G_Relation` link. The tag on the link can either be a `g_type` relation such as “S” (that represents a sort-of link) or a `g_label` if the link represents a predicate (Plink) in an RDF triple.

The third graph shows two different contexts for a relation: in the first, at the conceptual level, two functional object properties are linked by a “S” `G_Relation`; in the second, the factual level, ovals denote two resources linked by a `G_Relation`. where the user-chosen string “has wife” is the `g_label` of the relation.

2.3. *G-OWL Detailed Metamodel*

The G-OWL metamodel include the primitives of the language derived from the `G_Entity` and `G_Relation` abstract classes of the high-level metamodel that are presented in the following graph using the UML language. We also present here informally their OWL 2 set-theoretic interpretation as defined in the W3C Direct Semantic document [6].

2.3.1. Detailed Entity Metamodel

Fig.4 presents the main components of the `G_entity` metamodel. The `G_entity` is divided into four abstract subclasses: `G_SingleObject`, `G_Collection`, `G_Property`, and `G_Container`, which are further divided. Each abstract class encompasses the concrete G-OWL's constructors that are associated with one or more visual symbols of the language

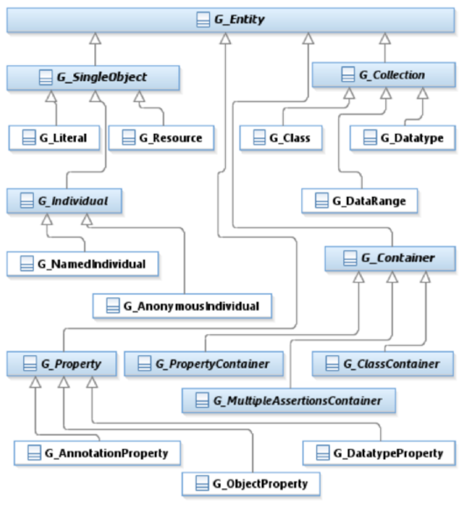


Fig. 4. Entity metamodel in UML notation

- *Classes* (`G_Class`) are interpreted semantically as sets grouping a number of member *Individuals* (definition by extension) or defined by some *Properties* that describe their attributes (definition by comprehension).
- *Individuals* (`G_Individual`) can be interpreted in OWL 2 as set members when they are declared as such.
- *Literals* (`G_Literal`) are members of a standard `G_Datatype` defined by W3C, such as strings, URI or IRI, integers or real numbers, or are more precisely defined `G_DataRange` that can serve as values of a Data Property.
- *Object properties* (`G_ObjectProperty`) are binary set relations between two classes, defined as a set grouping couples of *individuals*, the first taken in a first class (the domain) and the second in a second class (the range).
- *Data* properties (`G_DataProperty`) are also binary set relations grouping couples of two elements, the first taken in a class and the second in a class of literals of the same Datatype or Datarange.
- *Annotation Properties* (`G_AnnotationProperty`) are not part of the ontology but serve to describe the ontology or some of its components using reserved terms such as `owl:versionInfo`, `rdfs:label; rdfs:comment, rdfs:seeAlso` and `rdfs:isDefineBy`.

Containers are used in three different abstract ways:

- *Multiple Assertions Containers* group assertions of multiple relationships of equality/ equivalence or disjointness/differentiation about a list of individuals, classes, object or data properties.
- *Class Containers* construct new classes of the ontology. *Enumerations* construct a class by listing its individual members. *BooleanExpressions* construct classes composed by union, intersection or complement of already declared classes. *ClassRestrictions* construct new classes using an object property and its range. *Cardinality Restrictions* construct classes defined by the cardinality of their values given by an object or data property.
- *Property Containers* construct a new object property a functional composition of an ordered list of other object properties.

2.3.2. Detailed Relation Metamodel

As shown in Fig. , the derivatives of `G_Relation` in the G-OWL vocabulary allow the symbolization of notions of subsumption, equivalence, typology, etc. that assert relations between entities of Fig. 4.

The G-OWL visual syntax uses six relations or links, some using polysemy, in order to symbolize all of the axiomatic predicates of the OWL 2. It also uses two untyped relations to represent a predicate (or its negation) between a subject and an object resources in an RDF triple.

- The *instantiation link* (`ILink`) symbolizes the concept of typology used in the definition of RDF. The `ILink` is thus used to symbolise class membership of an individual (`rdf:type`).
- The *attribute link* (`ALink`) associates a Property with a Class. It is a typed link that is polysemic and polymorphic. From a class to a data or object property, it identifies the class as the (`rdfs:domain`)

of the property. From a property to a class or literal, it represents the (`rdfs:range`) of the property. The disambiguation of the link is built according to the orientation of the link, from the class to the property or from the property to the class. This use of polysemy on the A link represents better the semantics of a relation between classes, from domain to range.

- The *sort-of link* (`SLink`) symbolizes the ontological concept of Class or Property generalization. The link typed `SLink` is polymorphic, and the disambiguation of the link is evaluated according to whether the `SLink` is placed between two Classes, two ObjectProperties or two DataProperties.

- The *equivalence* (`ELink`) symbolizes the ontological concept of equivalence or equality. This link is also polymorphic, and its disambiguation is done according to whether it is placed between two Classes, ObjectProperties, DataProperties or Individuals. In the first three cases it represents an equivalence relation. In the last one, it represents an equality between two Individuals.

- The *inverse property* (`InvLink`) symbolizes the relation between an ObjectProperty and its inverse.

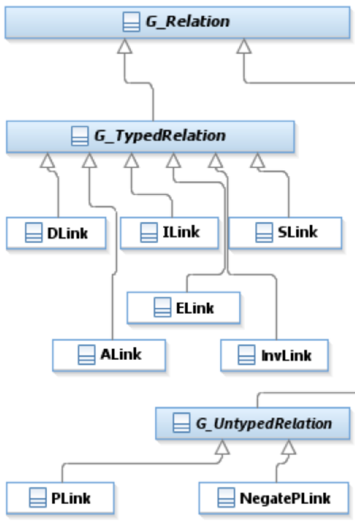


Fig. 5. Relations metamodel in UML notation

3. The G-OWL Visual Ontology Language

Based on the previous section, we will now present the concrete G-OWL visual ontology language as a representational system [63, 64]. Similarly to Sowa [65], we will use the Peirce’s meaning triangle (Fig. 6) to define the semiotic notions of *Object*, *Symbol*, *Concepts*, and the relations between these notions.

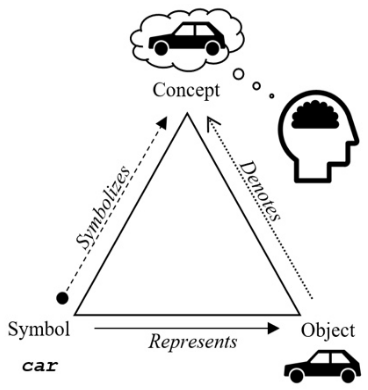


Fig. 6. Pierce Semiotic Triangle

3.1. *The G-OWL Semiotic Model*

In the semiotic model, the *Object* designates an observable and tangible world entity, or an abstract entity, delimited by a field of interest called the domain of discourse. The object of a domain of discourse may be material (e.g., an automobile, a house, etc.) or imaginary (e.g. a color, a process, another language etc.). The *Symbol* represents the *Object*. For example, the symbol "car" is a word of English that may represent the object (🚗) of the reality.

The vocabulary grouping the symbols and the rules of arrangement of the symbols (the grammar) compose the notation [66] of the representational system (or language).

In the upper part of the semiotic triangle shown in Fig. 6, the *Concept* designates an idea, a notion, an abstraction that a human has made about an object, in this case the concept of a car. The *Object* 🚗 denotes this object of a car while the *Symbol* symbolizes the concept and represents the object.

In our case, the objects being represented are immaterial, since they are defined in W3C documents that describe OWL 2 concrete syntax (Fig. 7). These objects are the components of an ontology, the terms (T-box), the relations (R-box) and the assertions or axioms (A-Box) about any domain of discourse.

Since OWL 2 is a subset of the First Order Logic, these objects denotes set-theoretic concepts that are described in the W3C Direct Semantics document [6]. The semantics of the G-OWL symbols will use the same set-theoretic interpretation of the corresponding OWL 2 concrete syntax.

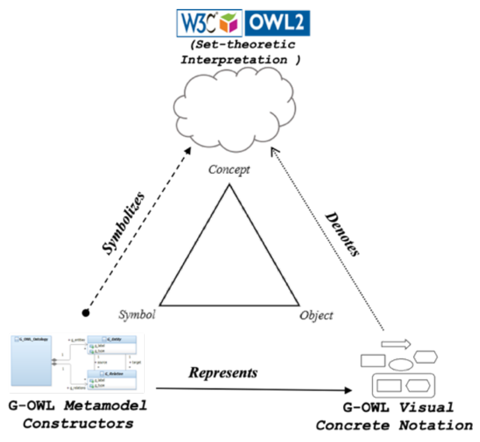

W3C OWL2
(Set-theoretic Interpretation)
Concept
Symbolizes
Denotes
Symbol
Object
Represents
G-OWL Metamodel Constructors
G-OWL Visual Concrete Notation

Fig. 7. G-OWL Semiotic Triangle

The objects of the OWL 2 concrete syntax can be expressed in several equivalent ways: RDF-XML, Turtle, Functional or Manchester Syntaxes. In this section we will use the Turtle concrete syntax to identify the objects that are to be represented using the G-OWL visual symbols which is another OWL 2 concrete visual notation.

These visual symbols are represented using the constructors that compose the G-OWL Metamodel of the language presented in section 2, Fig. 1-5.

3.2. *The G-OWL Visual Concrete Notation.*

In the following subsections, we present a number of concrete visual expressions of these constructors and of their combination that express the various components of an OWL 2 ontology.

Some of these concrete visual symbols will be presented together with their OWL 2 Turtle code equivalent and with the set-theoretic semantics they symbolize.

3.2.1. Basic G-OWL Concrete Visual Symbols

We start here by showing in Fig. 8 the basic entity and relation visual symbols available in the G-OWL visual language.

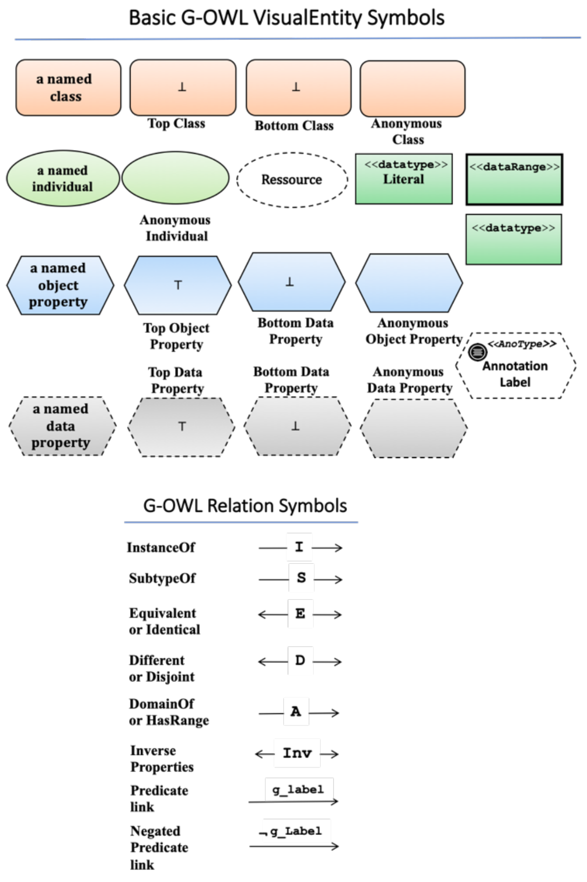


Fig. 8. G-OWL Basic Visual Symbols

In the first part of figure 8, the visual entities are built with the entity constructors presented in Fig. 4. The basic vocabulary of an ontology, classes, datatypes, individuals, literals, object and data properties is represented by corresponding visual objects such as rectangles, ovals, hexagons with solid or dotted borders.

For example, an `owl:dataProperty` can be declared using a dotted hexagon with a string label such as “hasNumberOfChildren”. According to the Direct Semantic interpretation mentioned earlier, an `owl:dataProperty` such as this one is interpreted as a set of couples in the cartesian product D x L where D is a class like “Persons”, the domain of the property, and L is its range of values from a certain dataRange class like “Integers from 0 to 15”.

In the second part of Fig. 8, the visual relations are built with the relation constructors presented in Fig. 5. In combination with entities, they will enable the construction of the more complex declarations, assertions or axioms that constitute an OWL ontology.

3.2.2. Axioms About Entity Relationships

The relations in Fig. 8 serve to establish the relationships between entities shown on Fig. 9. The “I” link serves to assert that an individual is a member in a class. The polymorphic S sort-of link asserts a “sort-of” relation between two classes, object or data property. The polymorphic E link asserts equivalence that two classes, object or data property are equivalent, or two individuals are identical. The polymorphic D link asserts that the entities are different or disjoint.

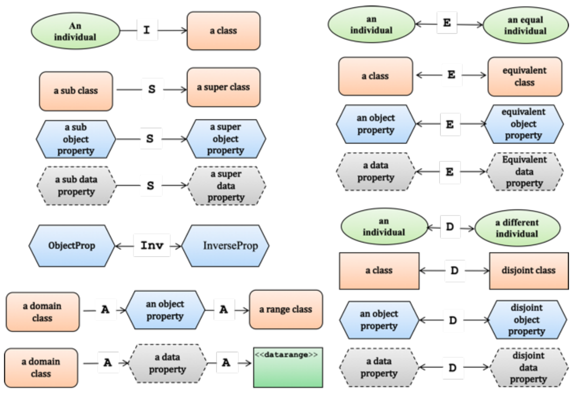


Fig. 9. Relationship assertions between basic visual entities.

Fig. 9 also displays the use of the “A” link to declare classes that are the domain and range of an object or a data property. According to the Direct Semantic interpretation, an `owl:ObjectProperty` is interpreted as a set of couples in the cartesian product D x R where D is a class, the domain of the property, and R is its range of values from another or the same class.

In Turtle concrete notation, an example with the object property hasWife would be expressed as:

```
:hasWife   rdfs:domain   :Man ;
           rdfs:range    :Woman .
```

The containers on Fig. 10 express in a condensed way multiple assertion about entity relationships of equivalence/identity, or disjointness/difference.

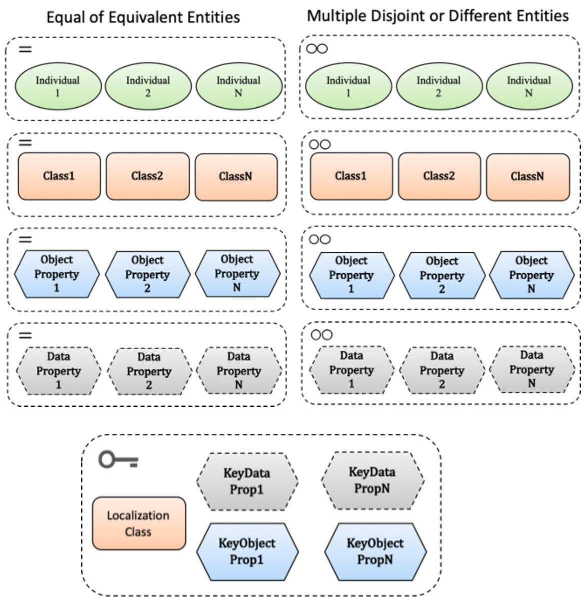


Fig. 10. Multiple an assertions about relations between entities

These containers have a type (“=” or “oo”) but no label because they are not ontology basic components like individuals, literals, classes or properties. They are not meant to be linked with other entities.

These entities are abbreviation of multiple assertions. For example, the second container in the right column express that the enclosed classes are pairwise disjoints. In set-theoretic semantics, it means that: Class i $\cap$ Class j = { }for all i and j.

In the Turtle OWL 2 concrete syntax, such a container is translated as:

```
[]  rdf:type   owl:AllDisjointClasses ;
       owl:members(Class_1,Class_2,…,Class_N)
```

In the second part of Fig. 10, one or more data or object properties can be declared as a key to identify uniquely the members of a certain localization class

3.2.3. Assertions About Property Type

Precise property types can be declared by adding a type symbol in the upper left part of an object and data property as shown on the first part Fig. 11.

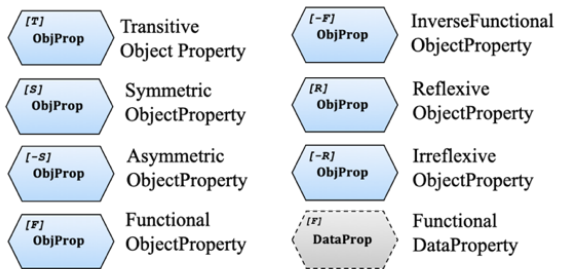


Fig. 11. Assertions about Property Types.

The corresponding Turtle concrete syntax for the first and last assertions are:

```
ObjProp   rdf:type  owl:TransitiveProperty .
DataProp  rdf:type  owl:FunctionalProperty .
```

and their set-theoretic semantic interpretation are respectively:

(x , y) ∈ ObjProp & (y , z) ∈ ObjProp implies (x , z)
(x , y) ∈ DataProp & (x , z) ∈ DataProp implies y = z

For object properties, these subtypes can be combined with some exceptions. A transitive property can be also symmetric or asymmetric, and also reflexive or irreflexive. A symmetric property cannot be asymmetric. A reflexive property cannot be irreflexive.

3.2.4. Classes Constructed by Boolean Operations or Enumeration

Unlike the examples of Fig. 10, container symbols are used most of the time to build new classes from individuals or other classes as shown on Fig. 12. The first container constructs a new class by enumerating its `owl:Individual` members. The other containers combine already declared *Classes by* using Boolean intersection, complement, union or disjoint union.

The last container for disjoint Union declares that a new class, named ClassLabel, groups all the individuals that are in at least one of the Class 1 to Class N but not in their pairwise intersection.

All these containers are classes that can be linked with other classes in the same way as simple classes, for examples with S or E links. They can also be linked with individuals by I links or with properties by A links as their domain and range.

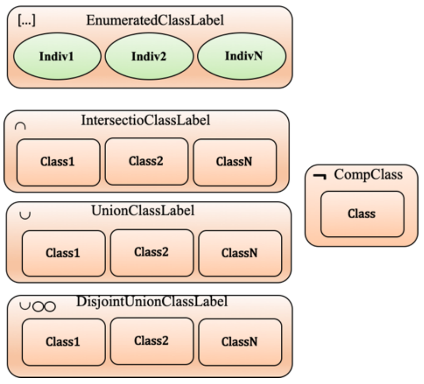


Fig. 12. Class Containers by Enumeration or Boolean Operations

3.2.5. Classes Defined by Property Restrictions

Besides enumerations and Boolean constructs, containers are also used to define new classes using property restrictions shown on Fig. 13. In the first two containers, the class named "ClassLabel" groups all the individuals that that have at least one (or all) of their values in the "ValueClass", of the property.

For Data properties, more than one property can be used to group all individual having at least one (or all) of their values in their combined DataRange.

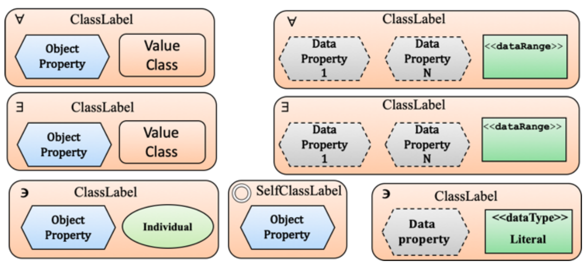


Fig. 13. Class Containers by property restrictions

The Self Class container groups all the members of the domain of the ObjectProperty that have a value in its range identical to this member.

In the two other two containers, the constructed class groups all the individuals that have a certain "Individual" or a certain "Literal" as their value by the "Object Property" or by the "Data Property".

The first two left-side containers have a set-theoretic meaning as sets of individuals (I).

$\{x \in I \mid \exists\ (x,y) \in \text{ObjectProperty} \land y \in \text{ValueClass}\}$

$\{x \in I \mid (x,y) \in \text{ObjectProperty implies } y \in \text{ValueClass}\}$

The second left-side container has the following Turtle equivalent for a definition of parenthood:

```
:Parent owl:equivalentClass [
       rdf:type            owl:Restriction ;
       owl:onProperty     :hasChild ;
       owl:someValuesFrom :Person ] .
```

3.2.6. *Classes Defined by Cardinality Restrictions*

Another way to define new classes by a property is to use containers for cardinality restriction such as the ones displayed in Fig. 14. Similar ones not displayed are possible using a data property for their definition.

These containers require the specification of an integer N, and also a class in the case of qualified cardinality, together the object property. The qualified cardinality containers construct the class named "ClassLabel" grouping all the individuals that have exactly, at least or at most N values in the class "aClass".

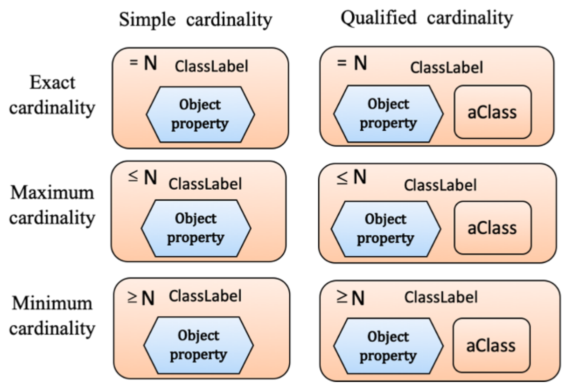


Fig. 14. Class containers by cardinality restrictions

With an integer value of N, the qualified cardinality have the following set-theoretic meaning:

$\{x \in I \mid \text{card}(\ \{y \in \text{aClass} : (x\ ,y\ ) \in \text{ObjectProperty}\}\ ) = N\}$
$\{x \in I \mid \text{card}(\ \{y \in \text{aClass}: (x\ ,y\ ) \in \text{ObjectProperty}\ \}\ ) \leq N\}$
$\{x \in I \mid \text{card}(\ \{y \in \text{aClass}: (x\ ,y\ ) \in \text{ObjectProperty}\ \}\ ) \geq N\}$

In the Turtle concrete syntax, a container with a definition of child would be translated as:

```
:Child rdf:type [rdf:type  owl:Restriction ;
 owl:minQualifiedCardinality 2^^xsd:nonNegativeInteger;
 owl:onProperty:hasParent; owl:onClass:Persons ] .
```

3.2.7. *Object Property Composition*

A third kind of container is displayed in Fig. 15. It is used to construct a new object property by the functional composition of an ordered list of already defined object properties.

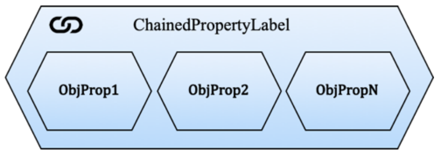


Fig. 15. Container Constructing a Composed Object Property

In a set-theoretic interpretation, the meaning of this container grouping three properties is as follows

$\{(x1,x4) \in \text{ChainedProp} \mid (x1,x2) \in \text{ObjProp1 and}$
$(x2,x3) \in \text{ObjProp2 and } (x3,x4) \in \text{ObjProp3}\ \}$

3.2.8. *DataRange and Datatypes*

The G-OWL language provides visuals ways (Fig. 16) to define precisely the available set of values used, for example, in the range of a dataProperty.

The simplest kind of a dataRange is any standard dataType from the following list:

- owl:real
- owl:rational
- xsd:decimal
- xsd:integer
- xsd:nonNegativeInteger
- xsd:nonPositiveInteger
- xsd:positiveInteger
- xsd:negativeInteger
- xsd:long

- xsd:int
- xsd:short
- xsd:byte
- xsd:unsignedLong
- xsd:unsignedInt
- xsd:unsignedShort
- xsd:unsignedByte

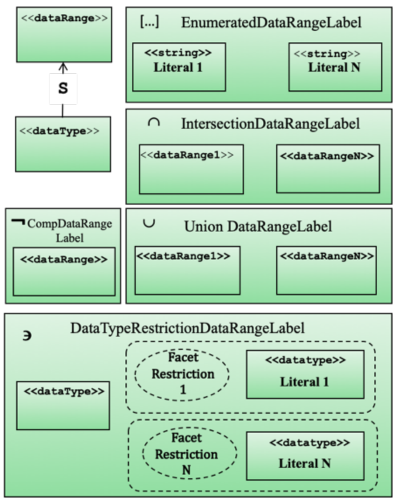


Fig. 16. Datatypes and DataRange composition

A dataRange can also be defined by enumerating a list of literal values from a dataType or by Boolean intersection, union or complement from already defined dataRanges.

A dataRange can also be defined by a data type restriction composed of any number of Facet restrictions. A Facet restriction is a couple of a (defined by a Resource IRI) together with its literal value of a certain data type. For example, the following functional description of a dataRange defined by a data type restriction on the Integer dataType contains exactly the integers 5 to 9.

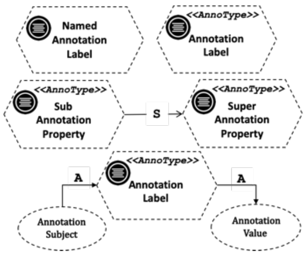
```
DatatypeRestriction  ( xsd:integer
        xsd:minInclusive "5"^^xsd:integer
        xsd:maxExclusive "10"^^xsd:integer )
```

3.2.9. Annotation properties

Annotation properties are provided in OWL 2 to add additional information about ontologies, entities, assertions or axioms. Even though they do not contribute to the semantics of the ontology, they are essential in practical Semantic Web projects.

G-OWL provides visual ways to define annotations properties that can be applied to any annotation subject with any kind of resource as its A domain, identified by its IRI. The annotation value of the property is another resource serving as its A range, identified as its IRI. Annotations properties can also be organized using the sub-property S link.

Fig. 17. Annotation properties

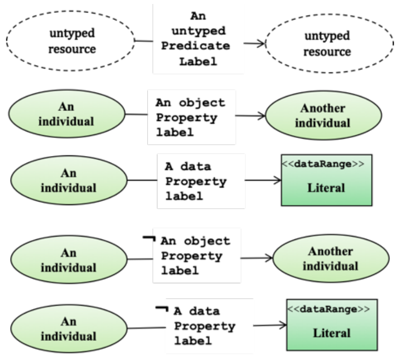


Named Annotation Label
<<AnnoType>> Annotation Label
<<AnnoType>> Sub Annotation Property
S
<<AnnoType>> Super Annotation Property
<<AnnoType>> Annotation Label
A
A
Annotation Subject
Annotation Value

Stereotype Syntax (AnnoType)	AnnoType Semantics
rdfs:label	readable name
rdfs:comment	comment
rdfs:seeAlso	additional info
rdfs:isDefinedBy	defining agent
owl:versionInfo	version information
owl:deprecated	deprecation
owl:backwardCompatibleWith	backward compatibility
owl:incompatibleWith	incompatibility
owl:priorVersion	prior version

A user can declare its own named annotation (with a user-defined label and a blank stereotype) or use one of the standard stereotypes <<AnnoType>> listed in the right part of Fig.17.

3.2.10. Resource Description Triples.



4. Fig. 18. RDF Assertions in G-OWL visual form.

G- OWL provides a visual way (Fig. 18) to assert facts in the form of RDF triples that can be processed by inference engines. RDF assertions can be made about any subject-object couple using a Plink with an untyped predicate label, or the Plink label of one of the typed object or data property label defined in the ontology. A predicate can also be negated by adding a ¬ symbol to its label, meaning the subject is not linked to the object by the predicate.

4.1. *Example of a G-OWL Ontology*

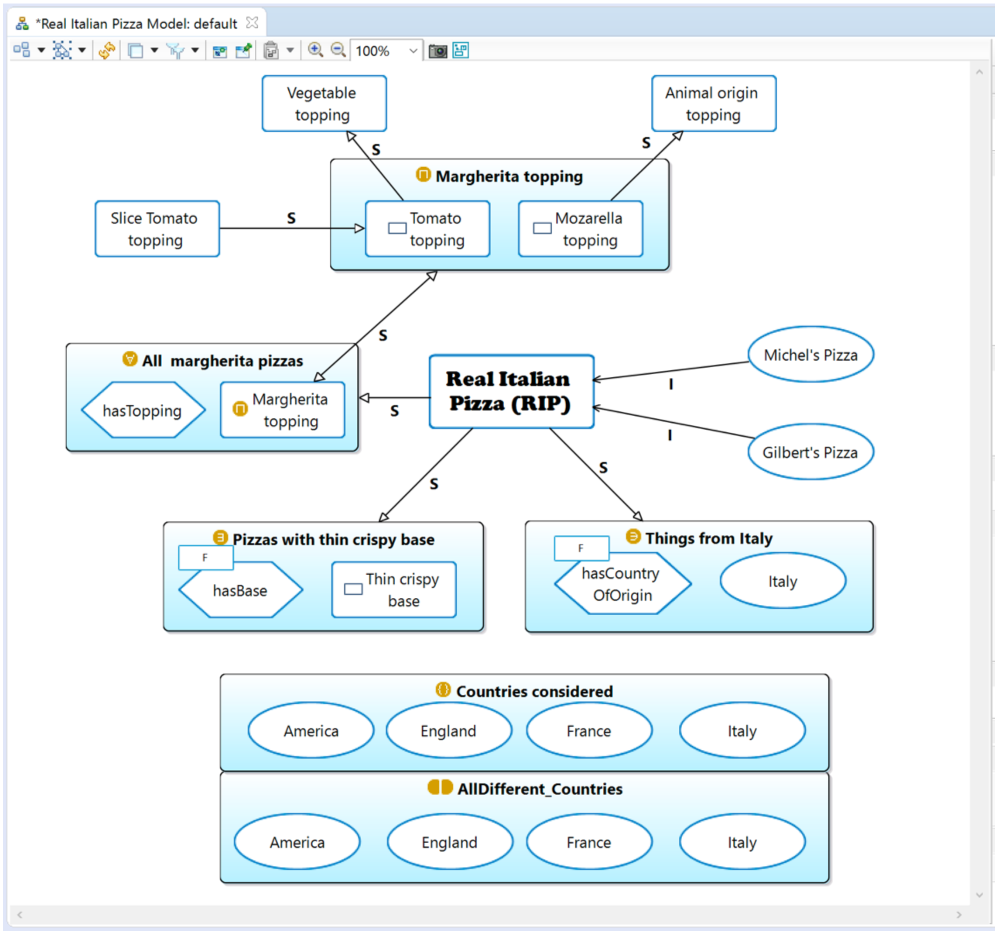


Fig. 19 – Part of an Ontology Expressed in the G-OWL Visual Language with the OntoCASE4G-OWL Modeler

We now complete the description of the G-OWL visual language by presenting part of a small ontology built with a prototype version of the OntoCase4G-OWL visual modeler for the G-OWL language[2].

Fig. 19 defines the concept of a Real Italian Pizzas (RIP), which is part of a larger ontology where various kind of pizzas from different countries are defined. The upper part of the model first defines the set of pizzas with a Margherita topping, as the intersection of pizzas that contain some kind of tomato topping and those with mozzarella topping.

This definition is reused in an OWL universal restriction container to assert that all RIP must contain a Margherita topping.

An OWL existential restriction asserts that some RIP can contain a thin crispy base. Another hasValue container asserts that a RIP has Italy as its country of origin. Two individual pizzas are declared being RIP pizzas using an instantiation (I) link.

[2] This visual modeler for the G-OWL language is actually in development. The colors used in the previous sections are not implemented. Also, the double Slink of figure 19 corresponds to the "E" equivalence or "same as" link of Fig. 8.

5. Visual Modeling Theory.

We now present in this section some of the theoretical background that has oriented the definition of the G-OWL language and that actually guides the development of its OntoCase4G-OWL modeler. It will also serve to evaluate the language in section 5.

5.1. *Communication Model*

Moody [66] presents a specialized version of Shannon's theory [67] of information communication that is relevant for the design of any language (Fig. 20). The process of transmitting information between a transmitter and a receiver starts with the encoding of a message in a certain notation by the transmitter.

In this model, noise represents a quality degradation of the information transmitted, resulting in misunderstandings the message by the receiver or in cognitive difficulty in coding the message by the sender.

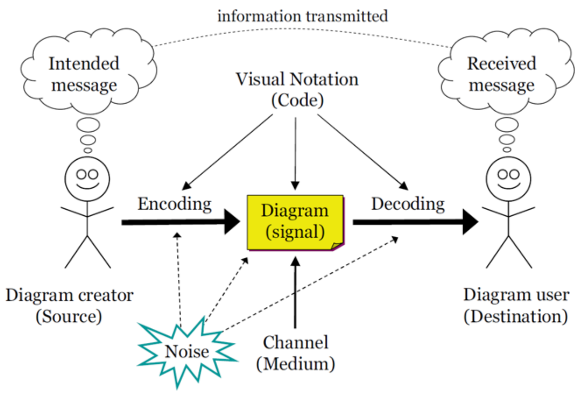


Fig. 20. Adaptation of Shannon's communication theory for visual communication (extracted from Moody [64])

In our case, the encoding is carried out using the visual notation of the G-OWL language, making it possible to create a diagram such as the one on Fig 19. The receiver uses her comprehension of the notation to decode the message.

We propose that a diagram created with a visual representation system can reduce the noise between the transmitter and the receiver compared to a text-based notation.

5.2. *Visual vs Textual Notation*

Larkin [68] notes that the fundamental difference between a "graphic" or visual notation and a textual notation is that a visual notation explicitly preserves the information of topological and geometric relationships between the components of a model. The visual notation also differs from the textual notation by the nature of the symbols [66] that composes its vocabulary as well as by the rules governing the interpretation of the symbols.

In a textual notation the symbols are displayed following a one-dimensional (linear) layout [68], sequentially aligned to form words. Words are also ruled by unidimensional and linear arrangements (grammatical rules) to form statements. The *unidimensional linearity* and the *sequential layout rules* are the two important notions that characterize a textual notation such as Turtle, RDF-XML, OWL-XML or the Functional and the Manchester syntaxes.

A visual notation uses visual symbols (geometric shape, icon, pictogram, etc.) includes in a visual vocabulary using color, size and position of geometric shapes together with rules for their visual arrangement [68, 69]. These rules are in our case *surface* rules for a 2D representation.

Hybrid notation (visual and textual) uses a vocabulary composed of both textual symbols and visual symbols that are governed by rules of textual and visual arrangement.

Although G-OWL contains some semantic aspects denoted by textual elements on the figures representing its entities or relations, there are no semantic aspects that are represented textually. Moreover, the G-OWL semantics is particularly sensitive to the shape, spatial position and arrangement of its visual symbols. This is why G-OWL is totally a visual notation.

5.3. *Moody's Physic of Notation Theory*

Moody's Physics of Notations Theory (PoNT) [66] is a systemic framework that has been used to evaluate, compare, improve and design visual notations in a wide variety of fields, including the Unified Modeling Language (UML) or the Business Process Modeling Notation (BPMN) [30, 70-72].

PoNT proposes nine principles as guidelines to design cognitively effective visual notations for human communication, in order to reduce information transmission noise. These principles were synthesized from theory and empirical evidence in a wide range of fields. They rest on an explicit theory of visual communication.

We will now present these nine principles: semiotic clarity, perceptual discriminability, semantic transparency, complexity management, cognitive integration, visual expressiveness, non-dual coding, graphic economy or parsimony, and cognitive fit. We intent to use these principles as a basis for the evaluation of a visual ontology language like G-OWL.

5.3.1. Principle of Semiotic Clarity and completeness

According to Goodman's theory of symbols [19], to achieve Semiotic Clarity and Completeness, a notation must have a one-to-one correspondence between each symbol and its referent concept.

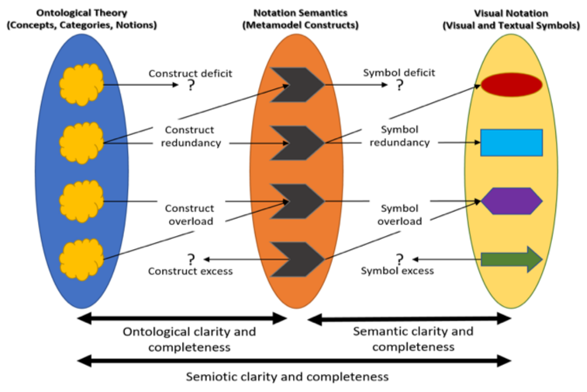


Fig. 21. Schematic View of the Principle on Semiotic Clarity and Completeness.

According to G-OWL Semiotic Triangle Fig. 7 adapted from Moody's definition [30], Fig. 18 relates three sets: the ontological concepts categories and notions of the *Ontological Theory*, the metamodel constructs of the notation's semantics, and the visual or textual symbols in the visual language or notation.

In our case, the first set groups the concept of OWL 2, the second contains the entities of Fig. 4 and the relations of Fig. 5 in the G-OWL metamodel, and the third contains the visual symbols contained in the G-OWL language presented on Fig. 8 to Fig. 18.

This figure illustrates a number of important features for a visual representation:

Deficit: *construct deficit* occurs when there are no constructors in the metamodel to symbolize a given ontological concept; *symbol deficit* occurs when there is no symbol to corresponds to a constructor in the metamodel.

Redundancy: *construct redundancy* occurs when there is more than one metamodel constructor to symbolize an ontological concept; *symbol redundancy* occurs when there is more than one symbol corresponds to a metamodel constructor.

Overload: *construct overload* occurs when a constructor is used to symbolize more than one ontological concept; *symbol overload* occurs when a symbol corresponds more than one metamodel constructor.

Excess: *construct excess* is a situation where there exists a constructor that is not associated with any ontological concept; *symbol excess* occurs when there are symbols that do not correspond to any of the semantic constructors.

The principle of Semiotic Clarity and Completeness ensures first' *completeness* of the notation so that each concept of the ontological theory has a symbol or a symbol pattern to represent it. Conversely, it assures that each symbol in the language serves to represent a concept of the ontological theory, in other words, has a meaning.

The principle of Semiotic Clarity and Completeness ensures maximum readability, eliminating the risk of ambiguity in the interpretation of symbols by a careful management of the deficit, redundancy, overload and excess of semiotic features.

5.3.2. Principles of Perceptual Discriminability and Visual Expressiveness

PoNT proposes here two principles. *Perceptual Discriminability* states that "Symbols should be clearly distinguishable from one another". *Visual Expressiveness* states that "Visual notations should use the full range of the 7 visual variables: position, size, value, texture, color, orientation and shape".

By increasing the visual distance between symbols, measured as the number of these visual variables, we favor a rapid understanding of the objects at stake in a visual model of an ontology and the relations between them, thus reducing the transmission noise.

5.3.3. Principle of Semantic Transparency

Semantic transparency [30] is the visual characteristic associated with a symbol to infer its meaning from its appearance. Semantic transparency is a measure bounded on one side by "semantic immediacy", that enables a novice reader to infer rapidly the meaning of a symbol, and on the other side, by the "semantic negativity", which intuitively induces an opposite meaning to that designated by the symbol.

5.3.4. Principle of Graphic Parsimony

The principle of *Graphic parsimony* defined by Miller [39] limits the number of distinct symbols used in a notation, for example by grouping them by categories or types.

Several empirical studies for the development of semi-formal notations (see sections 1.3 and 1.4), or in the interpretation of complex diagrams [73], show that there is a direct relationship between the number of symbols in a notation and the cognitive load associated with its use in activities such as diagram reading, notation learning or modeling with the notation. The risk of cognitive overload [40] increases if the number of symbols reaches or exceeds the *channel capacity* [39] of the person using the visual notation. Generally, a limit of seven (7) different symbols of each type should be a maximum.

5.3.5. Principle of Non-dual or Total Visual Coding

The principle of *Non-dual coding* states that "Notations should use text to complement (not replace) graphics". Textual annotations should be used as notes (to facilitate understanding) alongside graphics so that the ontology remains totally visual. This is a principle we apply in the OntoCase4G-OWL modeler for the G-OWL language.

In a previous study (in press), we examined eight different visual notations for OWL 2. We found out that most of them were not totally visual. For example OWLGrEd [22, 23] is a tool that provides good a UML style graphical notation for OWL 2, but where object and data properties are Manchester OWL 2 textual notations put on links between UML classes to represent properties, or as attributes within classes. Another example is WebVOWL [21] that covers visually many of OWL 2 notions but where restrictions or Boolean constructs are not displayed visually.

5.3.6. Principles of Complexity Management and Cognitive Integration.

Here PoNT provides again two principles. *Complexity Management* states that "Notations should include explicit mechanisms for dealing with complexity, such as modularization and hierarchy (abstraction)". *Cognitive Integration* states that "Notations should integrate information between separate diagrams".

Modularization of a large ontology can be achieved by dividing large ontologies into cognitively and perceptually manageable parts. Cognitive integration mechanisms refer to conceptual integration (summarization and visual momentum) and perceptual integration (signposting, orientation and navigation map). This principle is respected in the OntoCASE4G-OWL [60] modeler by the multiple canvas facility provided by the Eclipse IDE.

5.3.7. Principle of Cognitive Fit

The principle of Cognitive Fit states that "Different visual dialects should be used for different tasks and audiences". Notations should use different dialects for communicating with experts vs. novices. Visual notations for OWL 2 should be addressed primarily to content experts and ontology modelers. Computer scientists will in general rely mostly on XML textual representation of ontology constructs, using a visual representation for overviews. A notation that deserves the first group will also be a Cognitive Fit for the second if an interactives translation between the visual and a textual translation is made available at all times.

6. G-OWL'S Comparative Evaluation

In this section, we provide a systematic evaluation of the G-OWL language through a comparison (Fig. 22) with a popular visual language provided by Top Braid Composer [18]. Its two versions, the Diagram Editor (TBC-Diagram) and the Visual Graph Editor (TBC-Graph), are representatives of two kinds of visual representation notations. TBC-Diagram is an example of a UML-like notation, while TBC-Graph aims at a totally visual ontology representation.

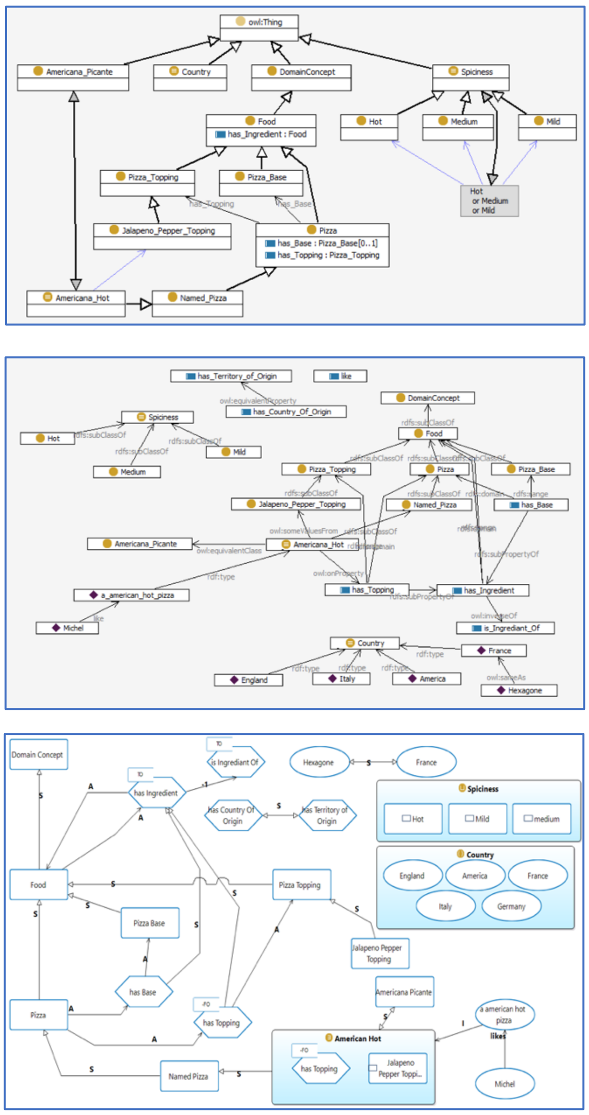


Fig. 22. An ontology in G-OWL, TBC-Diagram and TBC-Graph

6.1. *Global comparison and Semiotic Clarity*

Fig. 22 presents an overview of the same ontology respectively in TBC-Diagram, TBC-Graph and G-OWL notations.

The TBC-Diagram (a UML-like language) model is very compact, the entities are distinctly represented, the properties are treated as class attributes represented as texts within the classes, and the number of relationship types is minimal. We can also distinguish domain predicates very well versus ontological predicates (hierarchy and equivalence).

The TBC-Graph (a partially visual language) presents the elements of the A-BOX, the T-BOX and the R-BOX in the same diagram. We notice that the visual aspect of this graph-oriented symbolization makes it difficult to distinguish the different types of entities and relations.

The two TBC visualization syntax do not respect the Semiotic Clarity and Completeness principles totally. In TBC-Diagram, the ontological concepts of restriction, Boolean expression or grouping of individuals are not symbolized. Moreover, no element of the A-BOX is included in the model. In TBC-Graph, there is only one kind of link and all the kinds of properties are covered by putting OWL 2 textual expression directly on the links, making the representation dual, only partially visual, also making it difficult to distinguish visually between predicates from the domain and ontological relations.

The G-OWL visual syntax is on the contrary totally visual using 7 entity constructors and 7 relation constructors: ILink, SLink, ALink, Elink, Dlink, InvLink and Plink (asserted or negated).

Table 2: Entity Constructors and their Use in G-OWL

rectangle	literal, dataType and dataRange
rounded rectangle	class and classContainer
hexagon	objectProperty and chainPropertyContainer
dotted hexagon	dataProperty and annotationProperty
dotted rounded rectangle	multipleAssertionContainer
oval	individual
dotted oval	resource

The G-OWL Model is a totally visual language that allows the symbolization of elements of the T-BOX, R-BOX and A-BOX in the same model. It achieves Semiotic clarity and a complete visual symbolization of OWL by using two techniques called *Typology and Polymorphism.* These techniques have been widely used previously in the definition of the MOT language and modeling tools [43], G-OWL's predecessor, where their effectiveness in semi-formal communication processes have been widely demonstrated in a large number of projects in a variety of situations and uses. [28, 34, 36, 74, 75].

• *Typology* is the notion of assigning a predefined type to a symbol according to a categorization of constructors. The type is symbolized visually by a figure, an icon or a combination of characters. The disambiguation of the visual symbol is obtained by the interpretation of the type symbol as shown in the lower part of Fig. 23.

For example, in G-OWL, a property is represented by a hexagon. The addition of the symbol "F" to the hexagon type will identify a Functional Property, while the addition of the type "T" signifies that it is a Transitive Property. Here, the reduction of the number of symbols is achieved by applying *Symbolic redundancy* where one metamodel constructor (here the hexagon) corresponds in the language to many symbols of a same type.

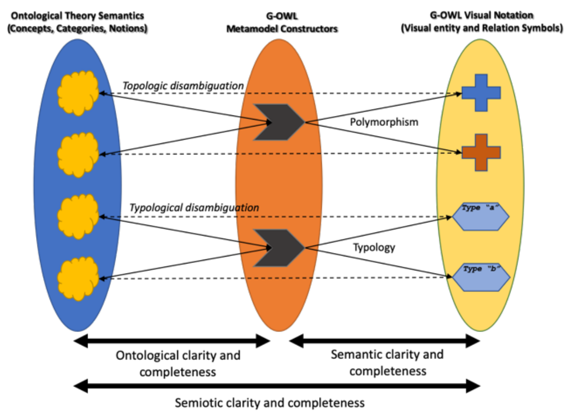

Ontological Theory Semantics (Concepts, Categories, Notions)
G-OWL Metamodel Constructors
G-OWL Visual Notation (Visual entity and Relation Symbols)
Topologic disambiguation
Polymorphism
Typological disambiguation
Typology
Type "a"
Type "b"
Ontological clarity and completeness
Semantic clarity and completeness
Semiotic clarity and completeness

Fig. 23 Typology and Topology disambiguation for Semiotic Clarity

• *Polymorphism* assigns to a given constructor several forms with very little semantic opacity. The disambiguation is assumed by the application of topological rules that takes into account the context of use of the symbol, as shown in the upper part of Fig. 23.

In G-OWL, polymorphism is used in particular for the notion of "specialization" or "subsumption" which is represented by the sort-of link "S" that can be placed between two Classes, two Object Properties, two Data Properties or two Annotation properties. Topological disambiguation is achieved by looking at the context of the link so that we do not need three different links. Here, the decrease in the number of symbols is achieved by the application of *Construct Overload* where one constructor (here link "S") serves to represent more than one ontological notion in OWL 2.

The use of these two techniques must assure the semiotic clarity and the completeness. Through typological or topologic disambiguation, a one-t-one correspondence is achieved between the first and the third spaces of Fig. 23, between the OWL 2 notions and the G-OWL entity and relation set of symbols.

6.2. *Graphic Parsimony*

The number constructors for entities and relationships respects the principle of Graphic Parsimony. Moreover, as mentioned in section 4, for each category of constructors, the number of sub-categories is also always less than 7, as prescribed by Miller's maximal *channel capacity*. The largest number, 7, is for the types of object properties: functional, inverse functional, transitive, symmetric, asymmetric, reflexive and antireflexive.

At first glance, the TBC-Graph syntax seems to be the most parsimonious since it contains only one entity, a rectangle and one relationship. All the nodes look almost the same (Fig. 24), but they are distinguished by 17 icons corresponding to those in the Manchester syntax [10] used in Protégé and four kinds of links. It does not respect the principle of parsimony, since the number of different nodes far exceeds the number of 7. Also, there is no polysemy of links that would facilitate the distinction between a specialization link between classes or properties, an instantiation link between an individual and a class or other kind of links.

The TBC-Diagram metamodel also consists of two generic symbols, the rectangle and the link. The rectangle encapsulates the properties similarly to class attributes in UML notation. Similarly, to TBC-Graph, the TBC-Diagram has a list of 16 rectangles distinguished by an icon presented in Fig. 24. There are also four types of relationships that serve to connect rectangles with each other. Some of these links are not related directly to ontology components.

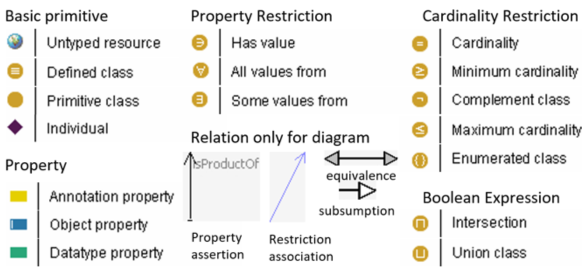


Fig. 24 TopBraid Composer Icons Legend of Graph and Diagram

6.3. *Perceptual Discriminability and Visual Expressiveness*

We have also retained the idea of using icons in the design of the G-OWL language, to help distinguish the four kind of restriction containers (Fig. 13), the five kinds of Boolean containers (Fig. 12), the Chain Property container (Fig. 15) and the various kinds of multiple assertion containers (Fig. 10).

But we have added visual expressiveness using different forms: rounded rectangles for class containers, hexagons for property contains and dotted rouned rectangles for multiple relationship containers. We have added also different colours to distinguish basic OWL 2 elements: green for individuals and literals, orange for classes, blue for object properties and grey for data properties.

In TBC-diagram and TBC-graph, as in many proposals for a visual ontology language, all the nodes look the same (rectangles). The addition of icons is useful but insufficient to help users grasp rapidly the structure and the main components of an ontology. This is evident in the graphs of Fig. 22.

6.4. *Semantic Transparency*

In the following sections, we will particularly consider to what extent do the three visual representations respect the important principle of Semantic Transparency prescribing that the appearance of visual symbols should help infer their meaning.

6.4.1. Taxonomy and Equivalence

In the first diagram of Fig. 25, we present a comparison of the three representations for a taxonomy of Classes and of for Object Properties. The second diagram compares these representations for the equivalence of Classes and Object Properties, and for the identity of Individuals.

For G-OWL, the hierarchy of ontological concepts is symbolized by a single symbol (`slink`) regardless of whether it is used to bind two classes, two Object Properties, or two data Properties. The polymorphism of the `slink` thus makes it possible to save three symbols while maintaining semantic transparency by

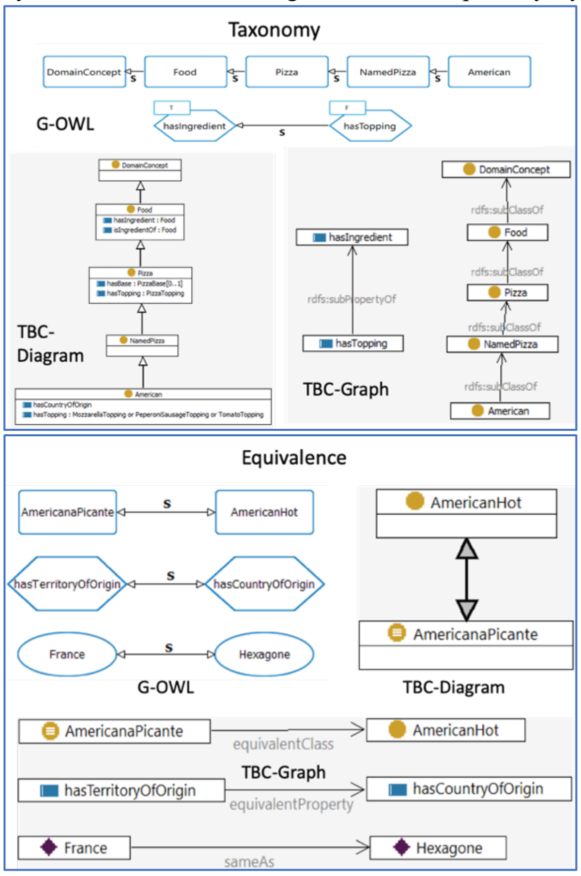


Fig. 25 Taxonomy and Equivalence Comparison

the choice of the label "S" to recall the semantic notion of the "sort of" or "subsumption" relationship.

For the TBC-Diagram, the class subsumption link has a high semantic transparency for UML users due to the reuse of the generalization relation commonly used in UML modeling. Property subsumption is not directly represented in this representation, and it must be deduced from the abstract rule that a subclass inherits properties of the superclass. Here, semantic transparency is notably lacking, especially for a beginner in ontological modeling or a non-UML user.

In the TBC-Graph, the distinction between class and object property subsumption is clearly identifiable by the typology of icons applied to each node, but still requires two different texts expressions on the link, which requires the use of two symbols to represent two concepts, at the expense of graphic parsimony.

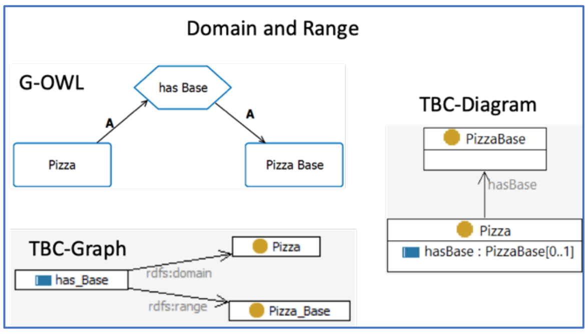
In the case of equivalence, the semantic transparency analysis is largely the same. In the TBC-Diagram, the visual syntax suffers from a constructor deficit for the expression of identity of individuals, while in the TBC-Graph, an additional symbol for the symbolization of the sameAs relation is necessary. G-OWL introduces the double-arrow symbol "S" for all three cases symbolizing the semantic of a double subsumption, corresponding to the semantic of the equivalence relationship.

6.4.2. Semantic Transparency for Signature, Instantiation and Domain Predicate

The semantic transparency associated with the signature (object and range) of an Object Property is presented in the first diagram of Fig. 26. In G-OWL, the polymorphic use of the `ALink` refers to the notion of attributes of a property. The direction of the arrows refers to the idea of the origin class (the domain) and the target class (the range) of a property, semantically seen as a directed relation between two sets, from the domain to the range. Therefore, in G-OWL, a single typed symbol is used to represent two ontological concepts, the domain and the range of property, simplifying the graphic complexity of the signature representation to a single symbol.

In the TBC-Diagram, the semantic transparency of the property is similar to that used in UML, thus facilitating reading for the UML expert. The representation still has symbolic redundancy since the "hasBase" property is found in two places and in two distinct visual contexts (in the Pizza class entity and in the relation between "Pizza" and "Pizza base"), thus inducing a certain semantic opacity since it is not trivial to deduce which of this representation refers to the range or domain of the property.

TBC-Graph centers the representation on the definition of the property since the links leave the property to go to the classes, without referring to a semantic of direction from domain to range. This representation has a greater graphic complexity than G-OWL since it requires two different links reproducing the textual notation.



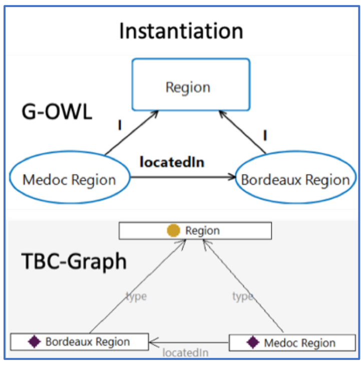


Fig. 26 Domain/Range and Instantiation Comparisons

In the case of instantiation, the second diagram of Fig. 26 shows that G-OWL, uses the `ILink` relation with the "I" symbol, referring to the predicate 'is-a' and the semantic of an "instanceOf" relationship.

Another strong semantic transparency is offered by the visual discrimination between the typed link that uses a standardized symbol like "I" and the untyped link "located in" that uses a domain specific symbol seen as a non-normalized string. This distinction facilitates the visual distinction between domain predicates and ontological predicates like “I”, “S” or “A”.

Like UML, the TBC-Diagram suffers from a deficit of constructors for the representation of the domain predicate and the predicate of instantiation.

The semantic transparency of TBC-Graph is again assured by its semiotic clarity using the “type” textual symbol. On the other hand, the semantic transparency is not helped by the fact that no visual clue facilitates the discrimination between the ontological predicate “type” and the domain predicate “located in”, both being represented by natural language strings.

6.4.3. Semantic Transparency for Restriction and Boolean Expression

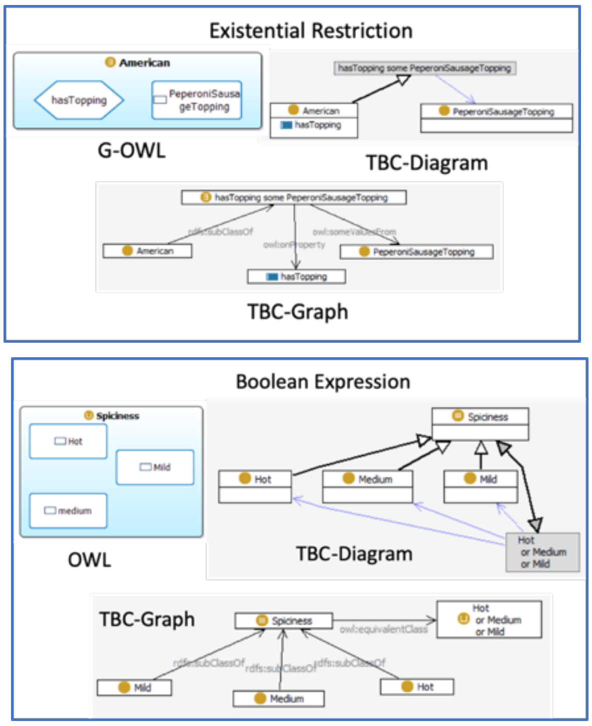


Fig. 27 Existential Restrictions and Boolean Expressions Comparisons

As shown by the two diagrams on Fig. 27, the semantic transparency of the restrictions or Boolean expressions differs considerably between the syntaxes.

For G-OWL, in both cases, the `g_container` visual symbol revisits the idea of grouping objects to compose a new visual object. In the case of existential, universal, has value restrictions (Fig. 13), Boolean expressions (Fig. 12) or cardinality restrictions (Fig. 14), the container represents a class that can be manipulated as a new singular visual entity. This semantic transparency allows the representation of complex ontological concepts and at the same time reduces graphic complexity avoiding the use of many links as in other representations.

The symbolic structure of the TBC-Diagram, inspired by the UML notation is handicapped by the fact that there is no UML notion or symbol that refers to the concept of restrictions or Boolean expressions. This lack of constructors requires the introduction of unconventional symbols to the UML standard which reduces the semantic transparency that might benefit users familiar with the UML notation. This add-on increases the graphic complexity by the introduction of a large number of symbols (relations and entities) to try to fit in OWL into the UML notation.

In the TBC-Graph, the concept of restriction and Boolean expression is also symbolized by the use of several links and several nodes. Nodes with complex labels (e.g. 'Hot or Medium or Mild') are also introduced that results in increasing the number of symbols necessary to represent the ontological concept. e.g.: five different typed symbols with four links to represent a simple Boolean expression.

7. Discussion

This section discusses general features of the G-OWL language based on three point of view:

1- comparative analysis of the diagrams on Fig. 22 to Fig. 27, and also of a previous study of eight visual ontology languages [3];

2- laboratory experiment on ontology modeling with 17 participants;

3- some preliminary functional software testing based on a version of the OntoCase4G-OWL modeler.

7.1. *Comparative Analysis of Visual Ontology Notations*

Based on the observations of the diagrams in the previous sections, some general observations can be made, both on UML-like notations like TBC-Diagram and Visual notations like TBC-Graph.

Gasevic *et al.* [61] provide an extensive summary of incompatibilities between UML and ontology languages. UML is based upon an object-oriented paradigm that provides many limitations for ontology visualization. For example, Ontology languages have the ability to construct classes using Boolean operations and quantifiers in property or cardinality restrictions. In UML, there is no corresponding primitives for these notions.

These and other differences between UML and Ontology languages enforce too many unnatural constructions. A good example is given by the graphs on Fig. 27. The first one is an example of an *Ontology Definition Metamodel* (ODM) complex model extracted from the *Object Modeling Group* OMG-ODM document [20, 76] based on the UML profile notation

This graph presents three OWL restrictions and one Boolean construction defining two types of "flowering plants", "azaleas" and "single colored azaleas". This last concept is particularly hard to decrypt since it is the intersection of two restrictions, one that uses a cardinality restriction to assert that these flowers have exactly one color, and the other that uses an hasValue restriction to assert it has a solid color pattern. A third existential restriction defines Azalea as flowers that have one of the "ASAColor pattern" as their bloom color pattern.

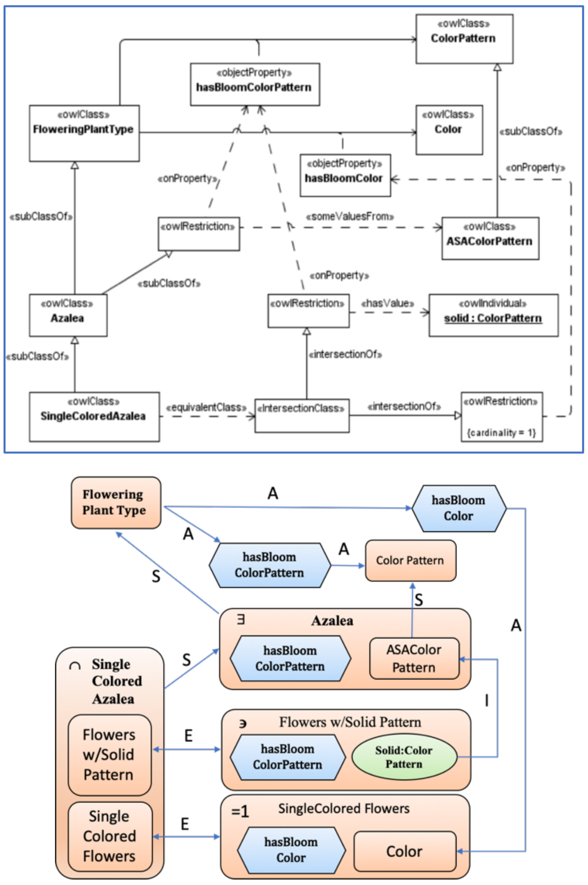


Fig. 28 An ODM and G-OWL Visual Comparison.

The G-OWL graph below is more human readable and semantically transparent, displaying clearly the restrictions and intersection operations as containers.

We have also added an instantiation link, not in the ODM graph, to assert that the individual "Solid: color pattern" in one of the ASA color patterns. Also, it is not necessary to use the links "onProperty", "hasValue", or "intersectionOf, since they are declared by their position in the containers. In ODM, these

[3] This study, "Designing and Communicating Ontologies Visually", is submitted for publication in the Semantic Web Journal.

terms are in fact copies from the syntax of the RDF-XML textual serialization, instead of referring to the OWL semantics. Also, the declaration of entities in stereotypes as being individuals, classes, object or data properties is unnecessary since they are given by their visual form and color.

Semantic Transparency is key 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. We believe that the differences between UML and Ontology languages enforce too many unnatural constructions.

Our study of four other visual syntaxes besides TBC-Graph, such as GrOWL [24], Graffoo [25] VOWL [21] or Graphol [26, 27] shows that they present, similarly to UML-like notations, some reuse of the textual syntax of OWL serializations, instead of symbolizing visually the OWL concept semantics, thus presenting a lower level of semantic transparency and a higher level of graphical complexity, especially when it comes to representing OWL restrictions or Boolean operations.

For example, in Graffoo, properties are identified by formulas on links. They are not visual objects by themselves, which precludes putting links between them to represent sub-properties or inverse properties. Textual axioms written in OWL 2 must then be added for those constructs not directly supported by any particular graphical element. Therefore, the notation is not Totally Visual.

VOWL proposes that information like disjointness, or types of properties like transitivity or symmetry should be listed textually in a sidebar instead of being presented visually. VOWL is not a *Totally Visual Modeling* editing tool. Some editing functionalities are introduced for OWL restriction or Boolean constructs, but these are not part of the VOWL visualization and must be displayed in a textual way in another window.

Graphol is based on the OWL 2 functional syntax and allows drawing ontologies in a completely visual way, including complex restriction or Boolean axioms. But the language uses similar visual symbols for basic ontology entities (classes, properties, individuals, datatypes), as well as for constructing operators such as restrictions or Boolean expressions. This defies the *Semantic Transparency* principle by having part of the visual symbols that are do not correspond to the language but to model construction operations.

7.2. *Laboratory Experiment Findings*

We have conducted a small laboratory experiment in order to evaluate the G-OWL language with users. It consisted in comparing the G-OWL language with other syntaxes commonly used in ontological engineering. A classification of the participants from companies and academia grouped them in three groups with different levels of expertise: 4 participants were classified at the novice level, 9 participating at the intermediate level and 4 participants at the expert level of ontology modeling

Inspired by the Cognitive walkthrough method [77], the G-OWL's degree of user-friendliness was measured for readability of an ontology. At this stage of the procedure, we presented a story text to the participant as well as six ontologies representing the story case in different syntaxes (G-OWL, TBC-Graph, OWL / XML, Turtle, Manchester, Functional). Subsequently, the participants gave their appreciation of the syntaxes by answering a questionnaire, setting a value between 1 (completely disagree) and 5 (completely agree) for each case:

1. I think the case is easy to read;
2. In a text, or a communication, I choose this case syntax to express my idea in an ontology;
3. I find that the case represents well the story text;
4. I consider that this case offers an intuitive reading;
5. I like reading the model of this case.

Table 3 summarizes the compiled results. Each cell has the percentage average of the persons who rated more than 3 for a question. The third column presents the best value of the textual syntaxes combined sets.

Even though this experiment will have to be extended in future work, we can conclude that for the interpretation of the ontology corresponding to the story text, the visual notations are more appreciated than the textual notations and that the G-OWL is slightly preferred to the representation in TBC-graph.

Table 3:
Readability Compiled Results

Question Numbers	G-OWL	TBC-Graph	Better Textual
1 easy to read	100%	76%	29%
2 in communication	94%	82%	19%
3 best for representation	100%	94%	69%
4 intuitive	100%	88%	30%
5 like reading	94%	76%	41%

Other uses of ontology language such has modeling, or deduction of new knowledge were also tested, giving us encouraging results. In after-experiment sharing, participants have indicated that for the deduction of new knowledge from a textual representation they usually pass through some kind of schematic visual representation step before making the inference of new knowledge. This is, for us, quite significant.

7.3. *Editing Software Functionalities Tested*

Several studies in visualization of ontologies mix the design of the syntax with the software tool that implements its use. In this research, we have presented the G-OWL visual language without its OntoCASE4G-OWL modeler. The main purpose was to assure the interoperability of the visual syntax by making it independent from a particular software implementation. Thus, the G-OWL language could support different software contexts such as Protégé.

The implementation of the actual OntoCASE4G-OWL software was an essential formal exercise aiming to validate the G-OWL metamodel. It made it possible to test a functional use of most of the OWL language for the production of ontologies that allowed comparisons with other visual syntaxes.

OntoCASE4G-OWL is an elaborated Eclipse-based application that operates the Sirius [78, 79] Framework for domain-specific graphical modeling language specification and implementation, the Eclipse Modeling Framework (EMF) [62] for data management of the models, and Apache Jena [79] for the Turtle - G-OWL serialisation and deserialization.

Many views are presented to the user by this modeler. A first one presents the entire ontology in a tree structure representation where components can be transferred to one or more model view (canvas), each accompanied by a palette to create ontological elements in G-OWL visual language. Each canvas presents a partial view of the ontology, either visual in G-OWL or textual in Turtle syntax, thus allowing the representation of a specific aspect of the ontology without changing its contents. Another view presents the properties associated with a selected item, which makes it possible to automatically generate graphic elements in the canvas.

The actual OntoCASE4G-OWL modeler covers almost all the OWL 2 semantics and of G-OWL language presented in this paper, but some specific elements (such as "hasKey" or cardinality restriction) have not yet been implemented as well as the colour code presented in section 3. However, given the wide range of OWL 2 semantics already covered, we believe that the extension of the modeler to the overall semantics of the OWL 2 will be straightforward.

8. Conclusion

In this paper, we presented the Graphical Web Ontology Language (G-OWL), a visual syntax that favors human readability and software modeling support for building semantic web ontologies.

The underlying hypothesis are that the use of polymorphism, typology, and polysemy, as well as the introduction of containers for central ontology-building operations, make it possible to reduce the number of symbols of the language while preserving the formal character and completeness of the language.

We have use Moody's Physics of Notations Theory (PoNT) principles to assert the following conclusions that the G-OWL visual representation system:

- is more human-readable than other OWL 2 concrete syntax, either textual or visual because it focuses on the set-theoretic semantics of OWL (semantic transparency) instead of the syntax of any textual serialization;
- contains a limited number of visual symbols to limit the cognitive load on the users (graphic parsimony) while covering all the semantic notions in the OWL 2 language;
- retains the formal character of OWL by the use of typology and topology disambiguation; for all G-OWL symbols or combinations of G-OWL symbols, there is only one ontological object of

OWL 2, and conversely, for each OWL 2 ontological object has a corresponding G-OWL symbol or combination of G-OWL symbols. (semiotic clarity preservation);

A distinctive characteristic of G-OWL is that its design principle aims to symbolize the semantics of OWL ontological concepts rather than the syntactic elements of a serialization notation as is the case for many other existing graphical syntaxes. Higher semantic transparency makes G-OWL a tool that guides the mind towards the representation of the meaning of elements of the domain of discourse rather than towards eventual codification issues pertaining to the syntax of the ontology.

In this sense, G-OWL is in line with the knowledge modeling tools that have been produced and validated during numerous previous researches on the MOT, MOTplus and G-MOT semi-formal modelers [33, 34, 41-43].

For the future, a scaling-up laboratory evaluation with a larger set of users is envisaged for G-OWL and for the on-going development of the OntoCASE4G-OWL modeler, in order to provide a mature tool for knowledge-intensive modeling projects.

At its origin, the G-OWL language was conceived with the aim of providing the knowledge engineer with a tool for eliciting and modeling ontologies for the semantic web. In the context of the evolution of Data Science and AI applications, we suggest that G-OWL will find its effectiveness in representing the vocabularies and schemas (R-BOX, T-BOX) that structure the Web of linked open data, while enabling assertions (A-BOX) to fuel the inference engines used in the Semantic Web applications.

Finally, it is hoped that the G-OWL language will serve as a basis for a future W3C recommendation for a concrete visual syntax for ontology modeling, well aligned with the supported textual concrete syntaxes.

Acknowledgments

This research project was conducted partly with the financial support of MITACS, the Canadian Industrial Research Grant Program and Hydro-Québec, a large hydro-electric company.

References

[1] G. Guizzardi, "On ontology, ontologies, conceptualizations, modeling languages, and (meta) models," *Frontiers in artificial intelligence and applications,* vol. 155, p. 18, 2007.

[2] W3C. (2012). *OWL 2 Web Ontology Language Primer (Second Edition)*. Available: http://www.w3.org/TR/owl2-primer/

[3] W3C. (2012). *OWL 2 web ontology language document overview*. Available: https://www.w3.org/TR/owl2-overview/

[4] W3C. (2014). *RDF 1.1 Concepts and Abstract Syntax*. Available: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/Overview.html

[5] W3C. (2014). *RDF Schema 1.1*. Available: http://www.w3.org/TR/2014/REC-rdf-schema-20140225/Overview.html

[6] I. Horrocks, B. Parsia, and U. Sattler. (2012, 2013-10-23). *OWL 2 Web Ontology Language Direct Semantics (Second Edition)*. Available: http://www.w3.org/TR/owl2-direct-semantics/

[7] W3C. (2012). *OWL 2 Web Ontology Language XML Serialization (Second Edition)*. Available: http://www.w3.org/TR/2012/REC-owl2-xml-serialization-20121211/

[8] W3C. (2014). *RDF 1.1 Turtle: Terse RDF Triple Language*. Available: http://www.w3.org/TR/2014/REC-turtle-20140225/Overview.html

[9] B. Motik, P. F. Patel-Schneider, B. Parsia, C. Bock, A. Fokoue, P. Haase*, et al.*, "OWL 2 web ontology language: Structural specification and functional-style syntax," *W3C recommendation,* vol. 27, p. 159, 2009.

[10] M. Horridge and P. F. Patel-Schneider, "OWL 2 web ontology language manchester syntax," *W3C Working Group Note,* 2009.

[11] M. Uschold and M. Gruninger, "Ontologies: Principles, Methods and Applications," *Knowledge Engineering Review,* vol. 11, pp. 93-136, June 1996 1996.

[12] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified modeling language reference manual, the*: Pearson Higher Education, 2004.

[13] O. F. A. Specification, "Business Process Modeling Notation Specification," ed: février, 2006.

[14] A. Ottensooser, A. Fekete, H. A. Reijers, J. Mendling, and C. Menictas, "Making sense of business process descriptions: An experimental comparison of graphical and textual notations," *Journal of Systems and Software,* vol. 85, pp. 596-606, 2012.

[15] R. Razali, C. F. Snook, M. R. Poppleton, P. W. Garratt, and R. J. Walters, "Experimental Comparison of the Comprehensibility of a UML-based Formal Specification versus a Textual One," 2007.

[16] Protégé Home Site. (2009, 2009-10-13). *Welcome to protégé*. Available: http://protege.stanford.edu/

[17] NeOn Project. (2009, 16 mars 2009). *Ontology design patterns*. Available: http://ontologydesignpatterns.org/wiki/Main_Page

[18] TopQuadrant. (2017, 26-09- 2017). *TopBraid Composer (TM)*. Available: https://www.topquadrant.com/tools/modeling-topbraid-composer-standard-edition/

[19] N. Goodman, Languages of art: An approach to a theory of symbols: Hackett publishing, 1968.

[20] OMG ODM. (2007, 26/05/2008). *Ontology Definition Metamodel: OMG Adopted Specification*. Available: http://www.omg.org/spec/ODM/1.0/Beta2/PDF/

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

[22] R. Liepinš, M. Grasmanis, and U. Bojars, "OWLGrEd ontology visualizer," in *Proceedings of the 2014 International Conference on Developers-Volume 1268*, 2014, pp. 37-42.

[23] J. Ovčiņņikova and K. Čerāns, "Advanced UML Style Visualization of OWL Ontologies."

[24] S. Krivov, R. Williams, and F. Villa, "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, 2007.

[25] R. Falco, A. Gangemi, S. Peroni, D. Shotton, and F. Vitali, "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.

[26] M. Console, D. Lembo, V. Santarelli, and D. F. Savo, "Graphical representation of OWL 2 ontologies through graphol," presented at the Proceedings of the 2014 International Conference on Posters &#38; Demonstrations Track - Volume 1272, Riva del Garda, Italy, 2014.

[27] D. Lembo, D. Pantaleone, V. Santarelli, and D. F. Savo, "Eddy: A Graphical Editor for OWL 2 Ontologies," in *IJCAI*, 2016, pp. 4252-4253.

[28] G. Paquette, "Graphical Ontology Modeling Language for Learning Environments," *Technology, Instruction, Cognition & Learning,* vol. 5, p. 36, 2007.

[29] G. Paquette and D. Rogozan, "Correspondance avec le langage graphique MOT-OWL et le langage des prédicats du premier ordre," LICEF, Montréal18 mars 2004, révision 22 mai 2006 2006.

[30] D. Moody and J. van Hillegersberg, "Evaluating the visual syntax of UML: An analysis of the cognitive effectiveness of the UML family of diagrams," in *International Conference on Software Language Engineering*, 2008, pp. 16-34.

[31] T. Buzan and B. Buzan, The Mind Map Book: How to Use Radiant Thinking to Maximize Your Brain's Untapped Potential: E P Dutton, 1994.

[32] J. D. Novak, D. B. Gowin, and J. B. Kahle, *Learning How to Learn*: Cambridge University Press, 1984.

[33] G. Paquette, Modélisation des connaissances et des compétences : un langage graphique pour concevoir et apprendre. Sainte-Foy: Presses de l'UQ, 2002.

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

[35] A. J. Cañas and J. D. Novak, "Concept Mapping Using CmapTools to Enhance Meaningful Learning," in *Knowledge Cartography: Software Tools and Mapping Techniques*, A. Okada, S. B. Shum, and T. Sherborne, Eds., ed London: Springer London, 2008, pp. 25-46.

[36] J. Basque and B. Pudelko, "Modeling for Learning," in *Visual Knowledge and Competency Modeling - From Informal Learning Models to Semantic Web Ontologies*, G. Paquette, Ed., ed Hershey, New York: IGI Global, 2010.

[37] B. M. Moon, R. R. Hoffman, and J. Novak, Applied Concept Mapping: Capturing, Analyzing, and Organizing Knowledge: CRC Press, 2011.

[38] J. Basque, G. Paquette, B. Pudelko, and M. Léonard, "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, Eds., ed London: Springer London, 2008, pp. 491-517.

[39] G. A. Miller, "The magical number seven, plus or minus two: some limits on our capacity for processing information," *Psychological review,* vol. 63, p. 81, 1956.

[40] P. Chandler and J. Sweller, "Cognitive load theory and the format of instruction," *Cognition and instruction,* vol. 8, pp. 293-332, 1991.

[41] G. Paquette, "La modélisation par objets typés - une méthode de représentation pour les systèmes d'apprentissage et d'aide à la tâche -," *Revue Sciences et techniques éducatives* vol. 3, pp. 9-42, 1996.

[42] M. Héon and G. Paquette. (2012). *Modélisation par objets typés*. Available: https://fr.wikipedia.org/wiki/Modélisation_par_objets_typés

[43] G. Paquette, M. Léonard, and K. Lundgren-Cayrol, "The MOT+ Visual Language For Knowledge-Based Instructional Design," LICEF-CIRTA Research Center and CICE Research Chair, Montréal2007.

[44] M. Héon, "Document de vision, Projet G-MOT, Composante B du projet PRIOWS," LICEF-TELUQ2010.

[45] G. Paquette and D. Rogozan, "Un processus de construction et d'évolution d'un système dirigé par ontologies," *Télé-université Montréal,* 2011.

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

[47] J. F. Sowa, "Conceptual Graph," in *Handbook on Architectures of Information Systems*, P. Bernus, G. Schmidt, and K. Mertins, Eds., ed: Springer-Verlag New York, Inc., 1999, p. 834.

[48] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modelling and Design.* . New Jersey, : Prentice Hall, 1991.

[49] G. T. Schreiber and H. Akkermans, *Knowledge engineering and management: the CommonKADS methodology*. Cambridge, MA, USA: MIT Press, 2000.

[50] C. K. West, J. A. Farmer, and P. M. Wolff, *Instructional Design, Implications from Cognitive Science.* . Boston: Allyn and Bacon, 1991.

[51] M. D. Merrill, Ed., *Principles of Instructional Design.* New Jersey: Educational Technology Publications, 1994, p.^pp. Pages.

[52] B. Inhelder and J. Piaget., *The growth of logical thinking from childhood to adolescence.* New York: Basic Books 1958.

[53] G. Paquette, *Instructional Engineering for Network-Based Learning.*: Pfeiffer/Wiley Publishing Co, 2003.

[54] G. Paquette, M. Léonard, I. de la Teja, and M. P. Dessaint, "Méthode d'ingénierie d'un système d'apprentissage MISA 4.0 Présentation de la méthode," *Une,* 2001.

[55] W3C. (2004). *OWL Web Ontology Language Reference*. Available: https://www.w3.org/TR/2004/REC-owl-ref-20040210/

[56] G. Paquette, I. Rosca, S. Mihaila, and A. Masmoudi, "TELOS: A service-oriented framework to support learning and knowledge management," in *E-Learning Networked Environments and Architectures*, ed: Springer, 2007, pp. 79-109.

[57] M. Héon and R. Nkambou, "G-OWL : Un langage de modélisation semi-formelle graphique pour la construction d'une ontologie dans la notation OWL," presented at the IC 2013: 24es Journées francophones d''Ingénierie des Connaissances, Cité Scientifique, Université Lille 1, Villeneuve d'Ascq, , 2013.

[58] M. Héon, Web sémantique et modélisation ontologique (avec G-OWL): Guide du développeur Java sous Eclipse, Collection Epsilon ed.: Editions ENI, 2014.

[59] 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 World Wide Web, Montréal, Québec, Canada, 2016.

[60] M. Héon, R. Nkambou, and M. Gaha, "OntoCASE4G-OWL: Towards an modeling tool for G-OWL a visual syntax for RDF/RDFS/OWL2," in *The 15 th International Semantic Web Conference DEMO-Session*, Kobe, Japan, 2016.

[61] D. Gašević, D. Djurić, and V. Devedžić, *Model Driven Architecture and Ontology Development*. New York, Inc.: Springer-Verlag, 2006.

[62] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework, Second EditionEMF: Eclipse Modeling Framework*, Second edition ed.: Addison Wesley Professional, 2008.

[63] V. Gérard, "A comprehensive theory of representation for mathematics education," *The Journal of Mathematical Behavior,* vol. 17, pp. 167-181, 1998.

[64] G. A. Goldin, "Representational systems, learning, and problem solving in mathematics," *The Journal of Mathematical Behavior,* vol. 17, pp. 137-165, 1998.

[65] J. F. Sowa, "Ontology, Metadata, and Semiotics," in *ICCS'2000*, Darmstadt, Germany, 2000, pp. 55-81.

[66] D. Moody, "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, 2009.

[67] C. E. Shannon and W. Weaver, *The mathematical theory of communication*: University of Illinois press, 1998.

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

[69] P. Irani, M. Tingley, and C. Ware, "Using perceptual syntax to enhance semantic content in diagrams," *IEEE Computer Graphics and Applications,* vol. 21, pp. 76-84, 2001.

[70] D. L. Moody, P. Heymans, and R. Matulevicius, "Improving the Effectiveness of Visual Representations in Requirements Engineering: An Evaluation of i* Visual Syntax," in *2009 17th IEEE International Requirements Engineering Conference*, 2009, pp. 171-180.

[71] G. Popescu and A. Wegmann, "Using the physics of notations theory to evaluate the visual notation of seam," in *2014 IEEE 16th Conference on Business Informatics*, 2014, pp. 166-173.

[72] N. Genon, P. Heymans, and D. Amyot, "Analysing the cognitive effectiveness of the BPMN 2.0 visual notation," in *International conference on software language engineering*, 2010, pp. 377-396.

[73] J. C. Nordbotten and M. E. Crosby, "The effect of graphic style on data model interpretation," *Information Systems Journal,* vol. 9, pp. 139-155, 1999.

[74] J. Basque, C. Imbeault, B. Pudelko, and M. Léonard, "Collaborative knowledge modeling between experts and novices: A strategy to support transfer of expertise in a organization," in *Conference on Concept Mapping*, 2004, pp. pp. 75-81.

[75] G. Paquette, M. Léonard, J. Basque, and B. Pudelko, "Modeling for Knowledge Management in Organizations," in *Visual Knowledge Modeling for Semantic Web Technologies: Models and Ontologies*, ed: IGI Global, 2010, pp. 393-413.

[76] D. Gašević, D. Djurić, and V. Devedžić, "The Ontology Definition Metamodel (ODM)," in *Model Driven Architecture and Ontology Development*, ed, 2006, pp. 181.

[77] C. Wharton, J. Rieman, C. Lewis, and P. Polson, "The cognitive walkthrough method: a practitioner's guide," in *Usability inspection methods*, N. Jakob and L. M. Robert, Eds., ed: John Wiley &amp; Sons, Inc., 1994, pp. 105-140.

[78] Obeo. (2016, 2016-07-04). *Eclipse Sirius: The easiest way to get your own Modeling Tool*. Available: https://www.eclipse.org/sirius/doc/

[79] V. Viyović, M. Maksimović, and B. Perisić, "Sirius: A rapid development of DSM graphical editor," in *IEEE 18th International Conference on Intelligent Engineering Systems INES 2014*, 2014, pp. 233-238.