# Drawing OWL 2 Ontologies with Eddy the Editor

Domenico Lembo[a,b], Daniele Pantaleone[a], Valerio Santarelli[a,b], Domenico Fabio Savo[a,b]
[a] *Dipartimento di Ingegneria Informatica, Automatica e Gestionale "A. Ruberti" Sapienza Università di Roma*
*⟨lastname⟩@dis.uniroma1.it*
[b] *OBDA Systems*
*⟨lastname⟩@obdasystems.com*

In this paper we introduce Eddy, a new open-source tool for the graphical editing of OWL 2 ontologies. Eddy is specifically designed for creating ontologies in GRAPHOL, a completely visual ontology language that is equivalent to OWL 2. Thus, in Eddy ontologies are easily drawn as diagrams, rather than written as sets of formulas, as commonly happens in popular ontology design and engineering environments. This makes Eddy particularly suited for usage by people who are more familiar with diagramatic languages for conceptual modeling rather than with typical ontology formalisms, as is often required in non-academic and industrial contexts. Eddy provides intuitive functionalities for specifying GRAPHOL diagrams, guarantees their syntactic correctness, and allows for exporting them in standard OWL 2 syntax. A user evaluation study we conducted shows that Eddy is perceived as an easy and intuitive tool for ontology specification.

Keywords: Ontology Design, Ontology Editors, Description Logics, OWL 2, Semantic Web

## 1. Introduction

In computer science, an ontology is commonly defined as a specification of a conceptualization, that is, a formal description of an abstract, simplified view of a certain portion or aspect of the world [27,28].

To provide *formal* representations of such views, ontologies have to be specified in a well-understood language, with mathematical-based syntax and semantics. Ontology languages are thus usually rooted in some kind of logic, which allows for automated reasoning over them, i.e., automatic inference of implicit knowledge from the ontology specification. Among the main proposals advanced over the years, we recall those relying on first-order logic [22,34] and its extensions [43], Description Logics (DLs) [7], frames [36, 26], production rules [49], Datalog and rule-based languages [11,8,35]. In particular, ontology languages based on DLs have become very popular, especially in the context of the Semantic Web [55]. Indeed, OWL 2 [31], which is the current release of the W3C standard Ontology Web Language, has its logical underpinning in the DL $\mathcal{SROIQ}(D)$ [15,33]. DL and OWL ontologies allow for representing the domain of interest in terms of concepts (a.k.a. classes), their attributes (a.k.a. data-properties), and binary relations between them (a.k.a. object-properties). Formal axioms involving those elements define the conceptualization by specifying the domain rules at an abstract level.

Another crucial property of an ontology is that it is *shared*, i.e., the representation it provides is agreed upon by all its users, so that it can act as a reference model across groups of people, communities, institutions, and applications [60]. Clearly, for an ontology to be shared, all the ontology users should be able to understand and exploit it, but this turns out to be not always easily achievable. In recent years ontologies have been introduced in several application contexts such as biomedicine, life and environmental sciences, e-commerce, e-government, cultural heritage [58]. Typically, people operating in these contexts are not able to easily interpret the logic-based formalisms through which an ontology is usually expressed and to completely exploit reasoning services over it, which instead they tend to consider simply a shared vocabulary.

Recently, we have experienced the above problem in various projects carried out in collaboration with industrial partners or public organizations (see, e.g, [5,6,54]). In these projects we have conducted an extensive ontology modeling activity and we have developed significantly large and complex ontologies, by working in collaboration with domain experts, who, in many cases, interacted with ontologies and languages like OWL for the first time. Our experience confirmed that modeling ontologies in contexts such as these can be very difficult if the representation is given in terms of logic formulas and by relying on tools designed for logicians. This turned out also to be an obstacle for transferring competences to our industrial counterparts, which wanted to take over the ontology development and management in the middle/long run, and to this aim asked for simple and user-friendly mechanisms for ontology representation.

To cope with the above problems, within our project activities we started the design and development of a new tool that could guarantee both formal specification of ontologies and their easy design and comprehension by our industrial counterpart. The result of our research investigation and implementation effort is Eddy[1], a novel ontology editing environment, which we present in this paper[2].

In Eddy ontologies are specified in GRAPHOL[3] [39, 42], a visual ontology language for OWL 2. A distinguishing feature of this language is that it allows for drawing ontologies in a completely diagrammatic way, i.e., GRAPHOL diagrams do not need to be annotated with formulas, even to capture complex axioms, which is instead usually required by other graphical formalisms for ontologies (e.g., [10,29,45]). Furthermore, GRAPHOL is proved to be equivalent to OWL 2, i.e., every OWL 2 ontology can be specified in GRAPHOL and vice-versa [42].

The basic elements of our language are borrowed from the Entity-Relationship (ER) model [14], and our tool offers services, such as diagramming and code generation functionalities, which are typical of environments for database and software design drawings. Thus, both GRAPHOL and Eddy are particularly suited for engineers, analysts, or designers in general who are used to describing requirements and designing systems through standard diagramatic conceptual modeling languages, like, e.g., ER or UML class diagrams [61].

At the same time, we notice that the syntax of GRAPHOL mimics that of DLs, in the sense that, as in DLs, a GRAPHOL ontology is a set of (graphical) inclusions between atomic or complex predicates, where the latter are constructed from the former through (graphical) operators. Thus, GRAPHOL has a very short learning curve for DL or OWL experts.

An in-depth user evaluation study has shown that GRAPHOL can be easily adopted by users knowledgeable in conceptual modeling but not necessarily in ontologies [42]. In these evaluation tests, GRAPHOL has been perceived as difficult as classical UML or ER diagrams for simple ontologies, and has been recognized as more intuitive and simple for ontologies that require the usage of complex axioms (e.g., that go beyond UML or ER), with respect to a language that requires to express them through formulas.

We also note that the above user evaluation study has been conducted without the support of a tool specifically tailored to GRAPHOL, since the aim of the test was to assess the effectiveness of the language, without being affected in this by any tool facility. However, the indications we have collected during this evaluation study have also highlighted the need of a dedicated environment for GRAPHOL, and some feed-backs we obtained have been useful to start the development of Eddy[4]. In particular, to help the user in the specification of GRAPHOL ontologies and support her in using logical constructs and operators, Eddy has been equipped with several functionalities, such as real-time checks on the syntactic correctness of the realized ontology, and automatic composition of some recurring ontology constructs (which is going to be extended in the future releases of the tool).

As already mentioned, the idea of using a graphical tool for knowledge representation is not new, and several proposals have been made during the years, in many areas of computer science, but none of them has succeeded in imposing itself as a reference graphical language for ontologies. Among them, the closest to our approach are those presenting UML-inspired languages and tools for the diagrammatic representation of OWL [10,21,29,45]. Differently from Eddy, none of them adopts a completely graphical language, but all require to complement the diagrammatic representation with textual formulas, in particular to encode complex OWL expressions. Other proposals that are not based on standard conceptual modeling languages, like [37,59], use structures such as graphs or concept diagrams to model ontologies. The notation used in these approaches is quite distant in nature from languages like ER or UML, and this turns out to be not completely suited in enterprise contexts. Also, these proposals have either been discontinued or are not able to completely capture OWL 2. We finally notice that popular ontology editing and engineering tools, such

[1] http://www.obdasystems.com/eddy

[2] The present article is an extended version of [40], demo paper presented at IJCAI 2016.

[3] http://www.obdasystems.com/graphol

[4] For a presentation of the GRAPHOL user evaluation study we refer the reader to [42].

as Protégé[5], TopBraid Composer[6], or OntoStudio[7], offer ontology visualization services, which provide overviews of portions of the ontology in some graphical format. However, they have little or no graphical editing features, and thus basically require the designers to be able to specify ontologies in terms of logic formulas.

In the rest of the paper, we first briefly describe the GRAPHOL language and provide an example of a GRAPHOL ontology. This is the subject of Section 2. Then, in Section 3, we present Eddy. In particular, we show the main functionalities offered by our tool for drawing GRAPHOL ontologies, and provide the algorithms it implements for ensuring that Eddy produces correct OWL 2 ontologies. Indeed, Eddy allows users to export GRAPHOL ontologies in standard OWL 2 syntax, which is a crucial functionality to enable easy interaction with other OWL-based ontology editors and with DL reasoners.

In Section 4, we discuss the results of a user evaluation study we conducted to verify the effectiveness of Eddy, in which we involved people from the industrial world with some skills in conceptual modeling and database design, but without experience in ontology construction or engineering. Our tests show that Eddy has been perceived as an easy tool for ontology specification. Then, in Section 5, we illustrate the ongoing and future directions for the development of Eddy, whereas in Section 6 we discuss the most relevant related work, and in particular recall the main features of most popular ontology editors. We finally conclude the paper in Section 7.

## 2. The Graphol language

In this section we provide an informal general overview of the GRAPHOL language, whereas for a complete description of its syntax and semantics we refer the reader to [42].

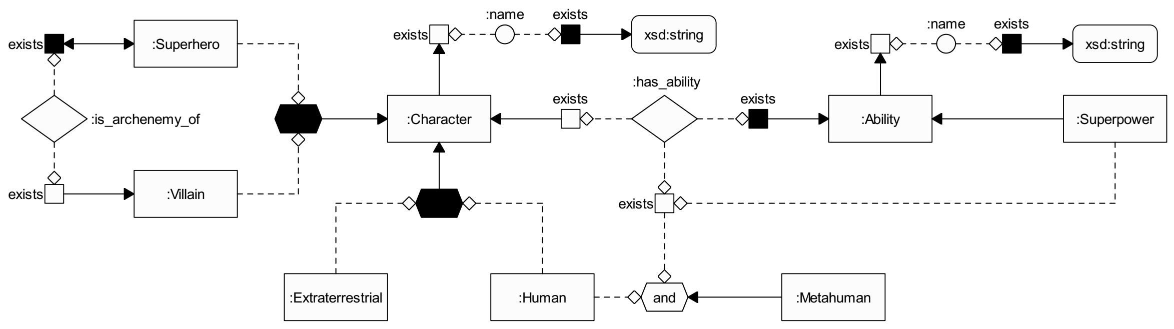
GRAPHOL is a graphical formalism which has an inherent formal semantics based on DLs and resembles ER diagrams, but which is in fact able to fully capture OWL 2. We begin by showing an example in the top part Figure 1, which models the world of comicbook characters and their abilities. In words, the ontology in the figure is saying that, among characters, there are Humans, Extraterrestrials, and Metahumans. Humans and Extraterrestrials are mutually disjoint, i.e., a human cannot be an extraterrestrial and vice-versa. Every character has a name, that is a string. Some characters are Villains, whereas others are Superheroes. A Superhero cannot be a Villain and has an archenemy. A character can have an Ability. Every ability has a name, which is a string, and some abilities are Superpowers. Finally, Metahumans are Humans with Superpowers. In the bottom part of Figure 1 we provide a representation of the ontology through OWL 2 axioms specified in Functional-style syntax [47]. We remark that the two representations are equivalent, i.e., it is possible to define a transformation function preserving the semantics of the ontology and produce the OWL encoding from the GRAPHOL diagram, and vice-versa. In particular, for this example we used Eddy to both draw the diagram and automatically produce the corresponding OWL Functional-style syntax specification.

One can see that GRAPHOL's representation of the main atomic predicates, i.e., the symbols constituting the alphabet of the ontology, is analogous to ER. Indeed, rectangles are used for atomic concepts (called entities in ER), which denote sets of objects, diamonds are used for atomic roles, which denote binary relations between objects (corresponding to binary relationships in ER), and circles are used for attributes, which denote relations between objects and values from value-domains, e.g., `string` and `integer`, represented in GRAPHOL with rounded rectangles.

We recall that in OWL (and DLs) complex *expressions* can be built from atomic expressions (i.e., the atomic predicates) inductively with operators. For example, the union of superheroes and villains, denoted in OWL as ObjectUnionOf(:Superhero :Villain), is a complex concept expression constructed with the ObjectUnionOf operator applied to :Superhero and :Villain. Then, statements on how predicates (and expressions) are related to one another are specified through axioms. The most general form of an axiom is the *inclusion* between expressions. Obviously inclusions can be specified only between expressions of the same kind, e.g., two concept or two role expressions.

The basic idea behind GRAPHOL is to represent the ontology as a set of inclusions, as in OWL 2, while preserving a graphical representation of it, given in terms of a directed graph. The nodes of the graph are atomic predicates or graphical operators, whereas the edges are solid or dashed arrows. The latter denote input to operators (as explained below), whereas the former denote inclusions.

[5]http://protege.stanford.edu/
[6]http://www.topquadrant.com/tools
[7]http://www.semafora-systems.com/en/products/ontostudio/



```
ObjectPropertyDomain(:is_archenemy_of :Villain)
InverseFunctionalObjectProperty(:is_archenemy_of)
SubClassOf(ObjectUnionOf(:Superhero :Villain) :Character)
SubClassOf(:Ability DataSomeValuesFrom(:name rdfs:Literal))
SubClassOf(ObjectUnionOf(:Extraterrestrial :Human) :Character)
SubClassOf(:Character DataSomeValuesFrom(:name rdfs:Literal))
ObjectPropertyRange(:is_archenemy_of :Superhero)
SubClassOf(:Superhero ObjectSomeValuesFrom(ObjectInverseOf(:is_archenemy_of) owl:Thing))
SubClassOf(:Metahuman ObjectIntersectionOf(:Human ObjectSomeValuesFrom(:has_ability :Superpower)))
DisjointClasses(:Extraterrestrial :Human)
DisjointClasses(:Superhero :Villain)
SubClassOf(:Superpower :Ability)
DataPropertyRange(:name xsd:string)
ObjectPropertyRange(:has_ability :Ability)
ObjectPropertyDomain(:has_ability :Character)
FunctionalObjectProperty(:is_archenemy_of)
FunctionalDataProperty(:name)
```

Fig. 1. Example of a GRAPHOL ontology and its OWL 2 representation.

For instance, in Figure 1, the inclusion assertion between :Superpower and :Ability, corresponding to the OWL 2 axiom SubClassOf(:Superpower :Ability), is captured by the arrow between these two atomic concepts. For simplicity, inclusion edges between the same expressions and with inverse directions are denoted with a single solid edge with an arrow on both ends, which in fact corresponds to an equivalence between expressions.

GRAPHOL expressions are either nodes representing predicates of the ontology (e.g., atomic concepts, atomic roles, etc.), or a combination of these nodes with nodes that represent operators. These latter ones in GRAPHOL are of two kinds. The first are labeled blank and solid boxes, used for constructing role or attribute restrictions on the domain and range, respectively. We recall that the domain (resp., range) indicates the first (resp., second) component of a role or of an attribute. Here the label represents the type of restriction. For example, *exists* is used for existential restriction, which denotes the set of objects that occur in the domain (resp., range) of a role, and $(x,y)$ is used for cardinality restriction, which denotes the set of objects in the domain (resp., range) of a role that occur in at least $x$ and at most $y$ links instantiating the role (similarly for attributes). In general, in GRAPHOL we can add as many blank and solid boxes as needed to predicate appropriately on the domain and range of a role or attribute. The other operators are hexagons, which can be solid hexagons, representing a disjoint union, or blank labeled hexagons. In particular, blank hexagons can be labeled *and*, *or*, *not*, *inv*, and *chain*, respectively representing intersection, (non-disjoint) union, complement, inverse of a role (i.e., a role with inverted domain and range), and composition of roles. Expressions are built by connecting these nodes with dashed directed edges ending with a small diamond (called input edges). For instance, in Figure 1, the domain of :has_ability is denoted via an existential restriction, i.e., through a dashed directed edge from the role :has_ability to a blank box labeled exists. In particular this expression is included through an inclusion edge in the concept :Character, which corresponds to the typing of the first component of the role :has_ability to :Character (i.e., only characters can have abilities). In OWL 2 this is specified as ObjectPropertyDomain(:has_ability :Character) (notice however that this can be specified in OWL Functional syntax also as SubClassOf(ObjectSomeValuesFrom(:has_ability owl:Thing) :Character), which is similar to the GRAPHOL specification).

For a more complicated example, let us look at the inclusion used to specify that metahumans are humans with superpowers. In OWL 2 this is given through the axiom SubClassOf(:Metahuman ObjectIntersectionOf(:Human ObjectSomeValues-

From(:has_ability :Superpower))), where ObjectSomeValuesFrom(:has_ability :Superpower) expresses the existential restriction on the domain of :has_ability, qualified to the concept :Superpower (which denotes the set of objects in the domain of :has_ability linked by this role to some instance of :Superpower). In GRAPHOL, we construct this qualified existential restriction by linking through dashed arrows both :has_ability and :Superpower to a blank node labeled exists. This complex expression is in turn given as input to an *and*-labeled hexagon, together with the concept :Human, to take their intersection. The solid arrow from the concept :Metahuman to the intersection node finally depicts the wanted inclusion.

We also note that Eddy uses a special representation to draw functionality for roles and attributes, or inverse functionalities for roles. We recall that a role is functional (resp., inverse functional) if every object in its domain (resp., range) can participate only in one link instantiating the role (similarly for attributes). In particular, to represent a functional role (resp., attribute), Eddy uses a role (resp., attribute) node with a double blank border, to represent an inverse functional role, it uses a role node with a double solid border, and to represent a role that is both functional and inverse functional, as is the case for :is_archenemy_of in the example, it uses a combination of a double blank and solid border. In OWL 2 this corresponds to the two axioms FunctionalObjectProperty(:is_archenemy_of) and InverseFunctionalObjectProperty(:is_archenemy_of).

## 3. Eddy

In this section we introduce Eddy, the editor for the design and visualization of GRAPHOL ontologies.

GRAPHOL has been and is currently being used in various industrial and academic projects by teams of ontology designers for the design and maintenance of ontologies (see, e.g., [6]). In the absence of a custom editor, GRAPHOL ontologies were specified with general-purpose tools for graph drawing. However, in order to exploit GRAPHOL for ontology design to the fullest of its potential, it became clear that a tool specifically tailored to GRAPHOL was needed. Furthermore, this was confirmed by evaluation tests [42] conducted on GRAPHOL. Indeed, while GRAPHOL proved to be easily understandable and received positive feedback, several difficulties were encountered during the editing tests, where an off-the-shelf editor for graphs was used, equipped with a special palette providing the GRAPHOL symbols. These experiences gave way to the development of Eddy.

While designing the system we have kept in mind the most widely-accepted recommendations and principles of usability for human-computer interaction [19], e.g., Ben Shneiderman's eight golden rules [57] and Jakob Nielsens ten usability heuristics [52]. For instance, we provide shortcuts for commonly-recurring tasks, support reversal of actions (undo), provide error handling and prevention, e.g., real-time syntactic validation of the ontology, display graphical and textual feedback for user actions, etc. Furthermore, to minimize the difficulty of adoption, the interface and default layout are similar to those of most popular commercial graphical editors, i.e., a central modeling canvas, surrounded by two sidebar areas where the user can access tools for editing, navigating, and inspecting the ontology.

Eddy is available as an open source project[8] and is designed around a modular architecture, where the core component is enriched by additional features through plugins, which fosters collaboration in extending Eddy through new functionalities.

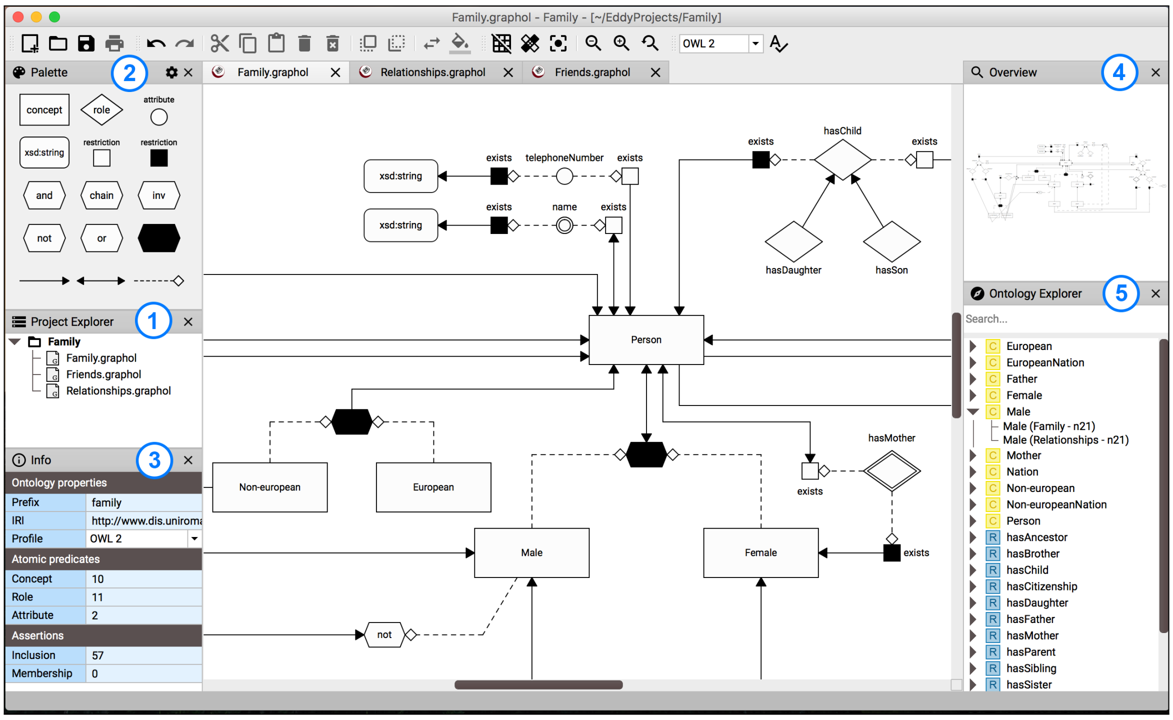
In the remainder of this section we will describe Eddy's architecture and its features.

### *3.1. Overview of the system*

The GUI, shown in Figure 2, is composed of the *Multiple Document Interface* (*MDI*)[9] area, which contains the modeling canvas, and two sidebar areas containing widgets for editing, navigating, and inspecting a GRAPHOL ontology. Each widget is a plugin which can be installed or uninstalled, and more can be added, guaranteeing flexibility to the system. The layout of the interface is completely customizable, and is maintained across multiple working sessions. Furthermore, the *toolbar* placed on top of the working area offers quick access to the majority of the operations provided by Eddy such as creating, opening or saving, scaling the canvas, and more. Eddy allows to create and manage multiple diagrams for a single ontology, for the purpose of easier visualization and comprehension, particularly in the case of large ontologies. An ontology predicate can occur as a node in different diagrams, and even more than once in the same diagram, while maintaining a single identity as predicate in the ontology. The user can access the diagrams of the on-

[8]http://www.obdasystems.com/eddy

[9]https://en.wikipedia.org/wiki/Multiple_document_interface



| Ontology properties | |
|---|---|
| Prefix | family |
| IRI | http://www.dis.uniroma |
| Profile | OWL 2 |
| **Atomic predicates** | |
| Concept | 10 |
| Role | 11 |
| Attribute | 2 |
| **Assertions** | |
| Inclusion | 57 |
| Membership | 0 |

Fig. 2. Eddy's user interface with some of the basic plug-ins.

tology from the *Project explorer* widget, (1) in the figure, each of which is displayed in a separate tab of the MDI. The *Palette* widget, (2) in the figure, provides all the nodes and edges in the GRAPHOL syntax, and is used to add elements to the currently active diagram through drag and drop or select-and-click. In Figure 2 we present a simplified version of the Palette, showing only the most widely-used symbols, introduced in the previous sections. The *Info* widget, (3) in the figure, shows the properties of the currently selected node. If no node is selected, it instead gives some general information of the ontology. Finally, the *Explorer* widget, (5) in the figure, provides a tree-view structure of the predicates of the ontology, each of which is identified by its label and a colored icon which reflects the type of the predicate. By expanding any predicate in the tree, Eddy also shows all the nodes in the diagram that represent it. Furthermore, it provides a search field to filter the predicates in the tree-view by showing only those matching the entered (partial) predicate name.

### 3.2. *Drawing functionalities*

Eddy offers a wide variety of drawing functionalities, from standard basic diagram editing features to more advanced features that are expressly tailored to the GRAPHOL language. The former comprise, among others, bending edges or moving edge anchor points, resizing nodes, moving or editing labels, and cutting/copying, pasting or dragging portions of diagrams.

GRAPHOL-specific drawing functionalities are provided by Eddy in order to aid users in rapidly performing commonly recurring tasks. These features are available through keyboard shortcuts and the contextual menus provided for each node in the diagram.

For instance, as shown in Figure 3, it is possible to choose the automatic composition of the domain or range of a role (resp., attribute), which results in a new blank or solid node labeled ‘exists’ being added in the diagram, linked to the selected role node through an appropriate dashed edge. Eddy also allows to switch the direction of the role, by automatically inverting the domain and range restriction nodes.

Moreover, it is possible to easily switch the label of a hexagon or a box from the contextual menu, which allows to instantly change the kind of operator represented by that node. This operation is syntactically controlled in real-time by the system: Eddy is provided with a context-aware mechanism that prevents the user from selecting an operator node type that would result in the composition of an invalid GRAPHOL expression (e.g., an intersection node whose inputs are concept

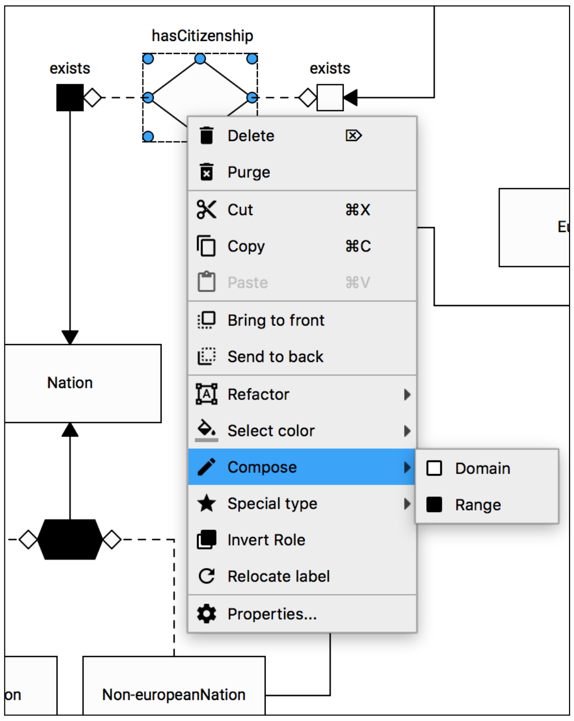
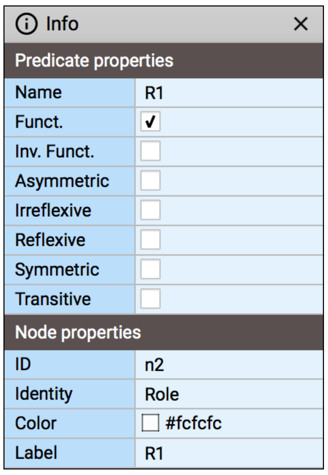


Fig. 3. Eddy's contextual menu for a role node.


Info

| Predicate properties | |
|---|---|
| Name | R1 |
| Funct. | ✓ |
| Inv. Funct. | |
| Asymmetric | |
| Irreflexive | |
| Reflexive | |
| Symmetric | |
| Transitive | |

| Node properties | |
|---|---|
| ID | n2 |
| Identity | Role |
| Color | #fcfcfc |
| Label | R1 |

Fig. 4. Eddy's Info widget for a role node.

expressions cannot be switched to a role inverse, role chain, or complement node). A similar mechanism is used to control the automatic switch of the direction of edges: this can be accomplished only if the result is syntactically correct in GRAPHOL.

Eddy is also equipped with an advanced feature for deleting GRAPHOL expressions from the diagram. The user can select this purge function on the sink node of an expression, and all nodes that form the expression and that are not involved in other inclusions or expressions will be deleted from the diagram.

### *3.3. Separation between predicate and node level*

As said, GRAPHOL allows to repeat the same predicate in the ontology through multiple predicate nodes. This is necessary because predicates that occur in numerous assertions in the ontology would otherwise be represented by a single node with numerous outgoing or incoming edges, resulting in layout and comprehension issues. This is compounded by the fact that in Eddy an ontology can be composed by several GRAPHOL diagrams, so this representation through multiple nodes can occur even across different diagrams.

Imagine now that the designer wants to change the name of a predicate, or to impose the functionality on a role or attribute, which, as said in Section 2, requires the node to be rendered in a special way (by using a double blank border). Of course these changes need to be repeated for every occurrence of a node representing the predicate that is being updated by the designer. From a practical point of view, having to perform these tasks singularly is obviously time consuming and error-prone. In fact, in Eddy this is not necessary, since our tool allows the designer to perform the above changes only once, since it automatically propagates them throughout the ontology. This is obtained by virtue of a logical separation realized in Eddy between the predicate level, the one concerning with predicates as logical symbols, and the node level, the one concerning the representations of predicates as nodes in a GRAPHOL diagrams. In general, seen at the predicate level a predicate is uniquely identified by its type and its name, whereas at the node level it might be represented by several nodes. The above "refactoring" tasks are thus performed at the predicate level, and propagated automatically at the node level.

In Eddy, this separation is put into effect in the *Info* widget, shown in Figure 4 for the case of a role node, where the properties of the selected node are separated into *predicate properties* and *node properties*. Predicate properties include the name of the predicate and, in the case of a role or attribute, its properties. Any change to these properties is reflected in all nodes in the diagram that represent the predicate. Node properties instead include the ID of the node and its label. Changes to node properties are performed on, and limited to, the specific node. For example, changing the la-

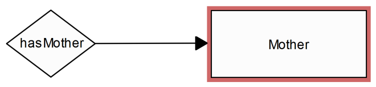


Fig. 5. Eddy's syntactic check: Invalid inclusion (between a role and a concept).

bel of the node will modify the name associated to the node only, and will not affect other nodes representing the same predicate (if any). After the renaming, two alternative situations are possible: if no other node of the same kind and with the same label exists in the diagram, the node represents a new ontology predicate, whereas it represents a new occurrence in the diagram of an already existing predicate, otherwise (in this case, if the node denotes a role, it inherits the predicate properties of the other nodes representing the same predicate).

### 3.4. *Syntactic validation functionalities*

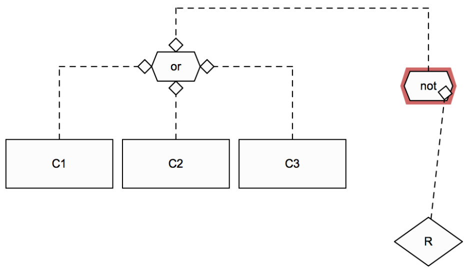


Fig. 6. Eddy's syntactic check: Invalid negative role expression.

Eddy is equipped with validation functionalities to aid the user during editing, assuring that the constructed ontology does not present syntax errors. Since the expressiveness of GRAPHOL coincides with that of OWL 2, this feature ensures that a GRAPHOL ontology can be correctly exported in a standard OWL 2 syntax.

This check takes place in real-time while the user is modifying the ontology. Given the characteristics of GRAPHOL, in Eddy a syntactic error can be produced only when drawing an edge between two nodes. Therefore, Eddy applies the check only when such an event takes place, and enables the change only if no errors are identified by the check.

Roughly, the check performed by Eddy consists in a simple lookup of the GRAPHOL syntactic rules, to verify whether they are violated by the update. This lookup is local, in the sense that Eddy needs to consider only the graphical elements involved in the update. More precisely, to establish whether an edge between two nodes can be added, Eddy needs to know the following information: the type of the edge to be inserted (i.e., whether it is an input or an inclusion edge); which are the source and target nodes between which the edge has to be traced (i.e., their shape and, if they are operator nodes, the label characterizing the kind of operator); the number and kinds of inputs to the target nodes. This last information is applicable only if the target node is an operator node and if the edge to be inserted is an input edge, and is needed to avoid exceeding the number and kind of inputs allowed for a certain operator node. Furthermore, Eddy needs to know the "identity" of the target and source nodes, i.e., whether they are concepts, roles, attributes, or value-domains.

Establishing the identity of a node is the complex aspect of the syntactic check. Indeed, while some nodes have a fixed identity (e.g., those that represent predicates of the ontology such as concept nodes and role nodes) others can change their identity according to the nodes they are linked to. We will refer to these nodes as *id-modifiable* nodes. For instance, a complement node, which is an id-modifiable node, can be used to build the complement of a role or the complement of a concept. In the first case the complement node assumes the identity of a role, while in the second, it assumes the identity of a concept. To denote the identity of a node we use the letters *C, R, A* or *V*, standing for concept, role, attribute, and value-domain, respectively. A combination of these letters indicates that the (id-modifiable) node can assume various identities, i.e., if the identity of a node is CV, it might become of identity C or V depending on how it will be linked to other nodes through input or inclusion edges. In fact, only the two combinations *CRAV* and *CV* are possible. The table in Figure 7 shows the identities that some of the most common nodes in Eddy's palette can assume, together with their default one, i.e., their identity if they are not linked to any other node of the ontology.

While the user is tracing an edge between two nodes, feedback regarding the correctness of the operation is displayed graphically by color-coding the target node when the mouse hovers over it. A valid connection is highlighted by surrounding the target node with a green frame, while an invalid connection, as shown in Figure 5, is highlighted by surrounding the target node with a red frame, and with a message on the status bar indicating why the connection is not valid. In this second scenario, the edge will not be added to the diagram.

| Node | Supported identities | Default identity |
|---|---|---|
| Concept | {Concept} | C |
| Role | {Role} | R |
| Attribute | {Attribute} | A |
| Value-domain | {Value-domain} | V |
| Domain restriction | {Concept} | C |
| Range restriction | {Concept, Value-domain} | CV |
| Complement | {Concept, Role, Attribute, Value-domain} | CRAV |
| Intersection | {Concept, Value-domain} | CV |
| Union / Disjoint union | {Concept, Value-domain} | CV |
| Role inverse | {Role} | R |
| Role chain | {Role} | R |

Fig. 7. Identities of most common GRAPHOL nodes.

Figure 5 showcases an extremely simple example of the behavior of Eddy's real-time syntactic validation, since both nodes that are being linked by the edge have a fixed identity. A more complex scenario is the one in which id-modifiable nodes are involved. Indeed, in this case to verify the compatibility of the nodes that the edge is connecting, their identity needs to be computed. For instance, consider the case in Figure 6, in which a user is attempting to connect a role R to a *not*-labeled operator node, which would lead to constructing the union of concept nodes and of the complement of a role node. Eddy must in this case understand that the identity of the complement node is *Concept*, which is not compatible with the identity of the role node, which is clearly *Role*, and the syntax allowed for inclusion edges in GRAPHOL.

In the following we describe (a slightly simplified version of) the technique used in Eddy for the syntactic check. Our tool in fact supports a refined and optimized version of the method described below.

To establish the identity of the nodes in a diagram, we use Algorithm ComputeNodeID (Algorithm 1). In the algorithm, $n$.ID denotes the identity of the node $n$ in the diagram $\mathcal{G}$, while defineID$(n, i)$ is a function that determines the identity that the id-modifiable node $n$ assumes when it is linked to a node with identity $i$. Establishing the identity depends on the GRAPHOL syntax, and thus the function defineID executes some lookups to the syntax to return the identity of $n$. For example, if $n$ is a union operator node, i.e., an or-labeled hexagon, and $i = C$, then defineID$(n, i)$ returns $C$. Notice that the function defineID can even return *CRAV* or *CV*. For example, if $n$ is a complement node with identity *CRAV*, and an input edge is traced from $n$ to an intersection node with identity $i = CV$, then defineID$(n, i)$ returns *CV* (indeed the complement node can now only become a concept or a value-domain node). We also

```
Input: a GRAPHOL diagram $\mathcal{G}$ with node identities;
       a node $n$ belonging to $\mathcal{G}$
begin
   $\mathcal{N} \leftarrow \emptyset$;
   $\mathcal{T} \leftarrow \emptyset$;
   $id \leftarrow n$.ID;
   if $id$ is different from $C, R, A, V$
   then $\mathcal{N} \leftarrow \mathcal{N} \cup \{n\}$;
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{n\}$;
   while $\mathcal{T} \neq \emptyset$ do
        select (and remove) some node $n_i$ from $\mathcal{T}$;
        foreach edge $(n_i, n_j)$ in $\mathcal{G}$ do
          if $n_j \notin \mathcal{N}$
          then if $n_j$.ID is different from $C, R, A, V$
               then $\mathcal{N} \leftarrow \mathcal{N} \cup \{n_j\}$;
                    $\mathcal{T} \leftarrow \mathcal{T} \cup \{n_j\}$;
          if defineID$(n_i, n_j$.ID$) < id$
          then $id \leftarrow$ defineID$(n_i, n_j$.ID$)$;
   foreach $n_i \in \mathcal{N}$ do
        $n_i$.ID $\leftarrow id$;
end
```

**Algorithm 1.** ComputeNodeID

establish a partial order between identities, stating that $CV < CRAV$, $C < CV$, $R < CV$, $A < CV$, and $V < CV$, whereas $C$, $R$, $A$, $V$ are incomparable one another.

In essence, ComputeNodeID takes in input a GRAPHOL diagram, annotated with node identities, and a node $n$, considers the GRAPHOL diagram as an undirected graph (and treats both input and inclusion nodes in the same way), and performs a visit of the portion of such graph that is reachable from the node $n$ by crossing only nodes with identity different from $C$, $R$, $A$, or $V$ (i.e., when the algorithm reaches a node $n$ with one such identity, it does not analyse other nodes reachable from $n$ that have not been previously visited). To carry out the visit, the algorithm uses two sets of nodes, $\mathcal{N}$ and $\mathcal{T}$, which are initially empty, and also initializes a variable $id$ to the current identity of the input node. Starting from node $n$, the visit is carried out, and all encountered nodes whose identity is still not established are collected in the set $\mathcal{N}$. When, during the visit, an edge $(n_i, n_j)$ is crossed, the function defineID$(n_i, n_j.\text{ID})$ is used to establish the identity of $n_i$ on the basis of $n_j$.ID. If such identity is lower, i.e., more specific, than an identity previously encountered, *id* is updated. This check is needed since the algorithm randomly selects the edge to visit at a certain iteration. For example, if we give, as input to ComputeNodeID, an intersection node having a union node and a concept as inputs, the algorithm might first select the edge connecting the intersection node to the concept, thus setting *id* to $C$. At the next iteration, the algorithm visits the edge connecting the intersection node to the union node, and finds that defineID returns $CV$ (since it only looks at these two nodes). Without comparing this identity with that previously computed we would overwrite the right identity for the nodes belonging to the visited portion of the diagram.

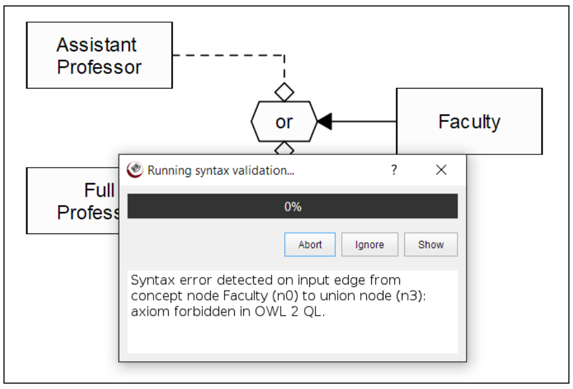
When the visit is completed, the identity of all nodes collected in $\mathcal{N}$ is set to the value of *id* (i.e., the input diagram $\mathcal{G}$ has been updated).

As already said, when the user tries to draw an edge between two nodes in a diagram, Eddy checks the correctness of this insertion, based on the GRAPHOL syntax and the identity of the two nodes, and adds the edge only if allowed by the syntax. Then, our tool executes ComputeNodeID to assign new identities to nodes, based on the presence of the new edge. In fact, if the identity of both the source and target node is different from *CRAV* or *CV*, there is no need to compute new identities, since they have been already established after a previous change in the diagram and cannot be modified by an edge insertion. Otherwise, it is sufficient to invoke ComputeNodeID only once, giving it as input either the source or the target node whose identity is not $C$, $R$, $A$, or $V$.

For our method to work it is also necessary to recompute identities of nodes when elements of the diagram are removed. In particular, when an edge is cancelled, ComputeNodeID is executed for both the source and the target node, i.e., it is invoked once with the source node as input, and once with the target node as input. If more elements are dropped together, ComputeNodeID is executed once for each node that is the source or the target of a removed edge (notice that in Eddy the removal of a node $n$ causes that all the edges incoming in and outgoing from $n$ are removed).

We remark that Eddy also supports the standard profiles of OWL 2[10]. These profiles are less expressive

[10]https://www.w3.org/TR/owl2-profiles/



Syntax error detected on input edge from concept node Faculty (n0) to union node (n3): axiom forbidden in OWL 2 QL.

Fig. 8. Pop-up window with feedback from the one-shot syntactic validation of the ontology.

fragments of OWL 2, and the user can select one of them from a drop-down menu in the toolbar. When one of the profiles is selected, the nodes that are outside of its expressiveness will not be selectable from the Palette, and the syntactic validation tool will be run with respect to the syntax of the chosen profile.

Besides real-time validation, Eddy also provides a one-shot syntactic validation feature, available through a button in the toolbar. A pop-up window, shown in Figure 8, notifies the user of the outcome of the test, and in case of a malformed expression or assertion, the user can choose to see the error in the appropriate diagram, ignore it and skip to the next one, or abort. Even though in principle a diagram designed in Eddy does not contain errors, situations as those described above can arise when the user wants to restrict to an OWL profile the expressiveness of an ontology initially written under the OWL 2 modality.

## 3.5. *Semantic validation functionalities*

Eddy is also equipped with semantic reasoning capabilities through the integration of an external OWL 2 reasoner, i.e., HermiT [24]. This allows the user to semantically validate the ontology by checking its consistency (i.e., whether it admits at least one model) or identifying unsatisfiable predicates (i.e., predicates that must have an empty interpretation in every model). If the ontology is inconsistent or one or more of its predicates are unsatisfiable, then Eddy provides the user with feedback regarding these malformations through their explanations, which are given both in graphical form by highlighting the Graphol axioms, and in textual form, through the OWL 2 axioms expressed in Functional Style syntax [47].

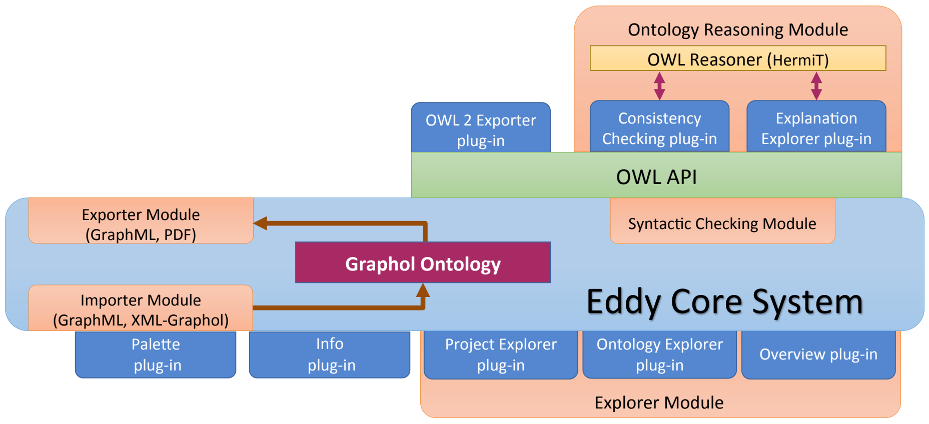

Ontology Reasoning Module
OWL Reasoner (HermiT)
OWL 2 Exporter plug-in
Consistency Checking plug-in
Explanation Explorer plug-in
OWL API
Exporter Module (GraphML, PDF)
Syntactic Checking Module
Graphol Ontology
Importer Module (GraphML, XML-Graphol)
Eddy Core System
Palette plug-in
Info plug-in
Project Explorer plug-in
Ontology Explorer plug-in
Overview plug-in
Explorer Module

Fig. 9. Eddy's system architecture.

We finally remark that the implementation of the semantic checks is based on the OWL API[11], a popular library for the creation, manipulation and serialization of OWL ontologies. Thus, despite the fact that in its current version Eddy uses the Hermit reasoner, any other OWL reasoner supporting explanation functionalities could be easily adopted in its place.

### 3.6. *Export and import functionalities*

By default, when saving GRAPHOL diagrams, Eddy adopts a proprietary XML-based file format, which we call XML-Graphol. In addition to the default format, Eddy provides the ability to export GRAPHOL diagrams in OWL 2 formats, to support interaction with third-party tools such as OWL 2 reasoners and editors like Protégé. Currently supported OWL 2 syntaxes are: Functional-Style syntax, Manchester OWL syntax [32], RDF/XML syntax for OWL [48] and Turtle syntax [9]. The user can either choose to export the whole GRAPHOL ontology into OWL 2, or to export only specific types of OWL 2 axioms. From a technical standpoint, the OWL 2 serialization of GRAPHOL ontologies has been performed with the help of the OWL API, which allows us to ensure compatibility between the OWL 2 serialization of GRAPHOL ontologies produced by Eddy and the majority of OWL 2 reasoners and editors.

Additionally, GRAPHOL diagrams can be exported in PDF or GraphML, an XML-based format for graphs, which allows to import them in general purpose graph editing tools, like yEd[12].

[11] http://owlapi.sourceforge.net

[12] https://www.yworks.com/products/yed

Import functionalities are also available to incorporate into the ontology GRAPHOL diagrams from external projects, which can be originally saved in XML-Graphol or GraphML.

### 3.7. *Design specifications and distribution*

Eddy is written in Python 3.4 [50] and makes use of the PyQt5 [3] python bindings for the cross-platform application framework Qt5 [4]. In order to translate GRAPHOL diagrams into an OWL 2 serialization, we adopted PyJNIus [2] to interface Python with the OWL API, which are implemented in Java. In Figure 9 we show Eddy's system architecture.

Eddy is licensed under the GNU General Public License v3 [1] and its source code is publicly available on GitHub[13]. We are currently providing a set of binary executables for Windows, Mac OS, and Linux, each including the redistributable version of the Oracle JVM (i.e., Oracle JRE 1.8), which is necessary to execute Eddy.

## 4. Experimental Evaluation

Many of the functionalities we have developed in Eddy derive from user interactions with GRAPHOL during the course of several industrial and academic projects, but also take into account the indications we have collected during a first user evaluation that we conducted on GRAPHOL, described in [42]. These experiences, while confirming GRAPHOL validity as a

[13] https://github.com/obdasystems/eddy.git

| | Age | Education | Ontology Experience (years) | Conceptual Modeling Knowledge | Ontology Knowledge |
|---|---|---|---|---|---|
| min | 36 | 1 | 0 | 3 | 1 |
| max | 64 | 3 | 5 | 5 | 5 |
| median | 51 | 2 | 1 | 4 | 2 |
| mean | 50.6 | 2.1 | 0.9 | 4.2 | 2.2 |
| st.dev. | 9.2 | 0.76 | 1.2 | 0.9 | 1.3 |

Fig. 10. Statistics of the participants: for Education, 1 = Bachelor Degree, 2 = Master's Degree, 3 = Ph.D; Conceptual Modeling and Ontology Knowledge are on a scale from 1 to 5, with 1 indicating no knowledge.

language for ontology design, have also highlighted the necessity of an ad-hoc editor for GRAPHOL.

Following the previous positive experiences of the user test conduced on GRAPHOL, we have carried out an evaluation on Eddy, in order to measure its usability for ontology editing by users who are already familiar with the GRAPHOL language, specifically for finding possible weaknesses in the user interaction with Eddy.

The evaluation test was conduced with 25 users, chosen among the attendees of advanced academic and industrial courses on semantic technologies, in which they acquired some familiarity with methodologies and languages for ontologies (including GRAPHOL). This number of participants is compliant with guidelines given in literature regarding best practices for usability tests for software tools, which indicate that five users is generally a sufficient number for qualitative tests, and roughly twenty is sufficient for a quantitative, or statistic, test [19,51]. All test participants come from the industrial world, and have some background in conceptual design, which is basically the know-how we assume for Eddy's users. In Figure 10 we recap some descriptive statistics about the participants regarding their age, highest completed education degree, their years of experience with ontologies, and their estimated knowledge of conceptual modeling and ontologies.

The structure of the test was the following:

1. *Introduction to Eddy*: users familiarized with Eddy by executing some guided simple tasks on an example ontology, e.g., creating and renaming concept nodes, undoing an operation, moving a node, drawing an inclusion edge between two nodes, copying a node, etc.
2. *Brief background questionnaire*: the participants had to answer a brief background questionnaire on their personal experience and expertise.
3. *Editing tasks*: each user was asked to carry out, through the Eddy editor, ten editing tasks on (a variant of) the Pizza ontology[14] specified in GRAPHOL. This ontology was chosen for its popularity among the Semantic Web community, and due to its simple and widely-understood domain. Each user was also asked to indicate the time it took to complete the task (in minutes), how clear it was to how to perform the task (on a scale from 0, worst, to 4, best), and how easy it was to carry out the task (on a scale from 0, worst, to 4, best).
4. *Ex-post survey*: the users were asked to fill out a brief survey to rate their experience with Eddy.

[14]http://protege.stanford.edu/ontologies/pizza/pizza.owl

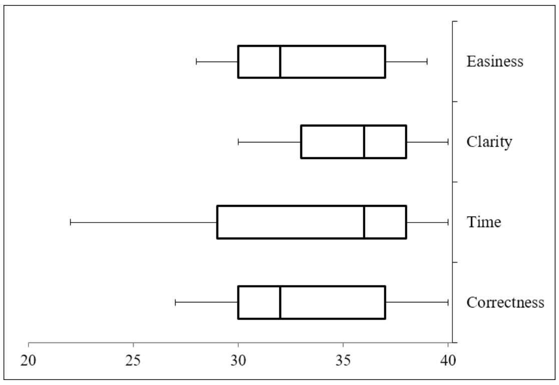


Fig. 11. Results of the Eddy user evaluation.

In Figure 11 we show a synthesis of the results. The correctness of each performed task was graded on a scale from 0 to 4, hence the maximum possible total score for correctness, as well as clarity and easiness, for each user was 40. Moreover, the predetermined benchmark average for time per task was set at 3.5 minutes. The figure shows the distribution of the total results in terms of, from top to bottom, perceived easiness of the tasks, perceived clarity of the tasks, time, and correctness. Each box plot shows the full range of variation, from minimum to maximum, indicated by the whiskers, the median value, and the likely range of variation, indicated by the two boxes, which represent the quartiles, i.e., the three points that divide the dataset into four groups, each comprising a fourth of the data. The left box is delimited by the first quartile (middle value between the smallest value and the median) and the median; the right box by the median and the third quartile (middle value between median and highest value).

Clearly, the high correctness scores and the low times per task, compared to the benchmark time, pro-

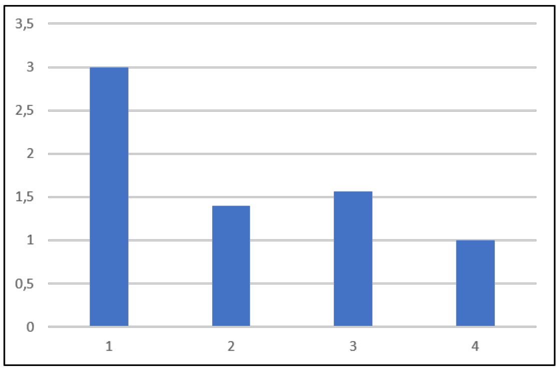


Fig. 12. Results of the ex-post survey.

duced by the users show a good ability in performing the required GRAPHOL modeling tasks through the editor. The high scores for clarity and easiness also indicate not only that the users were able to understand what they were required to do in each task, but also that Eddy allowed them to achieve their goal comfortably.

The average results for the mandatory questions in the ex-post survey are represented in Figure 12. We recall that for all four questions the scale is from 0 to 4, with 0 being the best possible value (note that question 1 involves the difficulty of the required editing tasks, and does not directly reflect the usability of Eddy). Clearly, these results reflect the fact that the experience with Eddy was absolutely positive for the participants to the test. Among the optional comments we collected, users particularly appreciated Eddy's contextual menu and keyboard shortcuts, since they allow to easily specify some properties, which are instead perceived as complicated in the GRAPHOL syntax. Suggestions for improvements included, for example, the possibility of importing OWL ontologies in Eddy, and providing more features for improving the alignment or distribution of the graphical elements in the diagrams.

Additional user feedback was obtained from the open questions in the post-test questionnaire and from a discussion with the users at the end of the test. This feedback indicated that the advanced drawing functionalities specifically tailored towards the GRAPHOL language were particularly helpful in quickly carrying out several of the tasks in the test.

## 5. Ongoing and Future Work

Eddy is currently under active development and we envision to extend its functionalities in several different directions. Some of these upgrades are currently being developed, and will be released shortly, others pose more significant challenges, and will require a more in-depth study. From the technical point of view, adding new features to Eddy is facilitated by its plugin-based architecture which guarantees extensibility and clear development direction.

A first upgrade we envision is the automatic drawing of some selected graphical patterns that correspond to classical conceptual modeling constructs, such as concept hierarchies, role typings, or mandatory participation of concepts into roles or attributes. Such functionality will enable the designer to select the pattern from a menu and thus have it automatically drawn in the ontology, ready for further customization.

An additional upgrade we envision for Eddy is the import of pre-existing OWL 2 ontologies. However, this problem is quite complex, and requires dealing with several issues. It will be necessary to develop a technique that processes the axioms that constitute an OWL 2 ontology, which do not provide any information regarding size and placement, and produce GRAPHOL diagrams. To obtain an optimal result in terms of the final layout, this technique must consider not only aspects such as minimal edge intersection or distances between nodes [18], but also some semantic criteria, which have yet to be devised, that allow to group together elements of the diagram that have a high semantic proximity. We feel that this issue, while particularly challenging, is very important, because it will allow us to achieve full interoperability with the OWL 2 language and its tools.

Another interesting aspects we will address is scalability which is particularly significant when dealing with the representation of very large ontologies. This is an aspect that must be considered in the perspective of user comprehension and of ontology manipulation. It is indeed unfeasible when dealing with very large ontologies that feature hundreds or thousands of concepts to include them all in a single graphical representation. In this regard incorporating functionalities for the support of ontology modularization [25,53] would be crucial.

Finally, in the long run, we are also investigating the possibility of expanding Eddy in other directions. For instance, we envision the possibility of extending Eddy in order to fully embrace the Ontology-based Data Management (OBDM) paradigm [44]. This calls for supporting the design of other elements of an OBDM specification, and not only the ontology. Indeed, in Eddy we would need to allow the users to link GRAPHOL ontologies with pre-existing data sources by means of mapping assertions. Therefore, we would need to provide an environment in which to specify such mappings, and also to validate them [38,41]. Furthermore, we would want users to be able to run queries over the generated ontologies and so coupling Eddy with external OBDM systems such as Mastro [13] and Ontop [12].

## 6. Related Work

Certainly, the most widely-used among ontology design environments is Protégé, a popular open-source ontology editor and knowledge base framework [23], created at the Stanford University. It supports a variety of formats for ontology development, such as OWL 2, RDF(S), and XML schema, and provides a plug-and-play environment that favors application development and the creation of new functionalities through plug-ins that can change both the behavior and the appearance of the system. Popular reasoners for DLs and OWL are available as plug-ins for Protégé, which thus allows for exploiting the standard inference services offered by such reasoners. Protégé does not provide graphical functionalities for ontology specification, but rather its approach is to support the designer in the writing of the textual logical formulas that constitute the ontology. In this respect, Protégé is completely different from Eddy, and it is thus mainly devoted to users who are expert in ontologies and formal languages, rather than to users without such skills but familiar with diagrammatic languages for conceptual modeling.

Other well-known ontology design and engineering environments have been developed within the Eclipse platform[15]. Among them, we mention TopBraid Composer[16], OntoStudio [63], and the NeOn toolkit [30]. The first two tools are commercial, whereas the last one is open-source. All of them allow for editing ontologies in various formats, such as OWL 2 or RDF(S), and provide functionalities for several ontology engineering activities, like management, reasoning, and collaborative development. None of them, however, enables for full graphical specification of ontologies, even though some aspects of the ontology can be edited or visualized through UML-like visual representations. For example, in NeOn this is realized through the OntoModel plug-in[17], whose development has however been dismissed to date.

[15]https://eclipse.org/

[16]http://www.topquadrant.com/products/TB_Composer.html

[17]http://neon-toolkit.org/wiki/OntoModel

OWLGrEd [45] and VOM (Visual Ontology Modeler)[18] are two other recent stand-alone tools for editing OWL 2 ontologies using UML. In particular, OWLGrEd adopts a variant of UML which requires to insert logical formulas in Manchester syntax [32] to represent complex OWL formulas that go beyond the expressiveness of UML. As already said, this is not necessary in GRAPHOL, the language used in Eddy, which is completely graphical, and in general compromises the intuitive understanding of the final ontology. VOM instead captures complex constructs in OWL by using UML stereotypes over classes and dependencies. This somehow flattens the graphical representation and hides its semantics in the meaning associated to the stereotypes. Also, both such tools do not provide a formal syntax of the graphical language they adopt (for a more detailed comparison between the language used in OWLGrEd and GRAPHOL we refer the reader to [42]).

Graffoo is a graphical notation for OWL ontologies, not based on UML [20]. In Graffoo ontologies are labeled graphs, which use several shapes for nodes and edges. Graffoo is specifically designed to capture OWL 2, but in order to do so, its visual representation is not completely graphical, as the constructs that are not directly supported by a native graphical element of the language are expressed through axioms in the OWL Manchester syntax, added as special nodes in the graph. Therefore the language suffers from the same difficulties discussed for previous tools regarding the need to embed logical formulas in the graphical representation. Graffoo currently does not come with an editing tool, but instead offers a palette for yEd, an open-source editors for graphs.

GrOWL [37] is a tool for visualizing and editing ontologies that, like Eddy, does not need to annotate the graphical representation with formulas. GrOWL, however, adopts a large number of symbols and distinguishes them also on the basis of their color and shading. Also, it uses DL notations for graph labels. Furthermore, the language did not evolve from OWL to OWL 2, and the project seems to have been discontinued.

While the above mentioned systems often attempt to provide some visual representation of ontologies, they are rarely successful in achieving a balance between the quantity of information that is provided to the user and the size and complexity of the overall representation, which is typically a two-dimensional graph.

[18]http://thematix.com/tools/vom/

Other tools and graphical notations instead focus uniquely on ontology visualization, and rely on different techniques to attempt to achieve this desired balance. The ontology models provided by these systems can be either two-dimensional or three-dimensional, and rely on visualization solutions such as multiple coordinated views [62], space-filling [56], degree of interest [16], and context focus [17]. Well-known examples of such systems are the OntoGraf[19] and OWLViz[20] plug-ins for Protégé. The former uses the layouts library for the Jambalaya plug-in[21] to provide interactive navigation of the relationships in an ontology through an incremental and dynamic graph-like representation. The latter provides a node-link representation for viewing and navigating class hierarchies, in which the nodes are classes, and the "is-a"-labeled links represent inclusion relationships between them. Whereas the above tools allow users to visualize only some aspects of the ontology, other works aim at a graphical rendering of the entire ontology. This is the case of VOWL [46], a visualization language for OWL implemented in two different tools, i.e., ProtégéVOWL, a Protégé plug-in, and WebVOWL, a stand-alone web application.

Because these visualization tools lack any kind of ontology editing functionalities, they fall slightly outside the specific focus of this work, so we will not discuss them further.

In conclusion, our brief survey on the most popular ontology design and engineering environments highlights that no currently available tool provides features and mechanisms for a completely graphical specification of OWL ontologies through a notation that is close to standard diagrammatic languages for conceptual modeling used in the enterprise context. Indeed, some tools do not provide at all functionalities for the graphical editing of ontologies (e.g., [23]), others give only a partial support to this, requiring to complement the graphical representation with formulas (e.g., [63,45]), others adopt notations that are far from conceptual languages such as ER or UML class diagrams (e.g., [37,20]), and not always have a clear relationship with OWL (e.g., [59]) (of course, some tools present various of the above limitations). We also remark that often no formalized syntax is provided for the graphical representation adopted in some tools (e.g., [45]), and that other, more formal, proposals have been dismissed to date (e.g, [37]). Finally, many systems only allow for a graphical (in general partial) visualization of ontologies (e.g., [46]) and not for the editing thereof. Our tool Eddy is thus specifically aimed to fill these gaps, by offering a completely graphical ontology editing environment, based on the usage of an ER-styled language, called GRAPHOL, which is equipped with formal syntax and semantics, and is equivalent to OWL 2 in terms of expressive power.

[19]http://protegewiki.stanford.edu/wiki/OntoGraf
[20]http://protegewiki.stanford.edu/wiki/OntoViz
[21]http://protegewiki.stanford.edu/wiki/Jambalaya

## 7. Conclusions

In this paper we have presented Eddy, an editor that provides advanced functionalities for designing syntactically and semantically correct OWL ontologies through their specification in GRAPHOL, a visual ontology language equivalent to OWL 2. Eddy is an effective tool for ontology development, as shown by the user evaluation study which we have presented in this paper, and is currently used in several real-world projects. Eddy is open-source and its latest version can be downloaded at http://www.obdasystems.com/eddy.

**Acknowledgments.** We wish to thank Maurizio Lenzerini and Giuseppe De Giacomo for many helpful suggestions they provided us for the development of Eddy.

## References

[1] Gnu General Public License (GPL). Gnu gpl version 3 Documentation, 2016. Available at http://www.gnu.org/licenses/licenses.html#GPL.

[2] Pyjnius. Pyjnius Documentation, 2016. Available at http://pyjnius.readthedocs.io/en/latest/.

[3] Python 3.4. Pyqt Documentation, 2016. Available at https://riverbankcomputing.com/software/pyqt/intro.

[4] Qt 5.7. Qt Documentation, 2016. Available at https://www.qt.io/.

[5] A. Amoroso, G. Esposito, D. Lembo, P. Urbano, and R. Vertucci. Ontology-based data integration with MASTRO-I for configuration and data management at SELEX Sistemi Integrati. In *Proc. of the 16th Ital. Conf. on Database Systems (SEBD)*, pages 81–92, 2008.

[6] N. Antonioli, F. Castanò, S. Coletta, S. Grossi, D. Lembo, M. Lenzerini, A. Poggi, E. Virardi, and P. Castracane. Ontology-based data management for the Italian public debt. In *Proc. of the 8th Int. Conf. on Formal Ontology in Information Systems (FOIS)*, pages 372–385, 2014.

[7] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2nd edition, 2007.

[8] J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. On rules with existential variables: Walking the decidability line. *Artificial Intelligence*, 175(9–10):1620–1654, 2011.

[9] D. Beckett, T. Berners-Lee, E. Prud'hommeaux, and G. Carothers. RDF 1.1 Turtle – Terse RDF Triple Language. W3C Recommendation, World Wide Web Consortium, Feb. 2014. Available at http://www.w3.org/TR/turtle/.

[10] S. Brockmans, R. Volz, A. Eberhart, and P. Löffler. Visual modeling of OWL DL ontologies using UML. In *Proc. of the 3rd Int. Semantic Web Conf. (ISWC)*, volume 3298 of *Lecture Notes in Computer Science*, pages 198–213. Springer, 2004.

[11] A. Calì, G. Gottlob, and T. Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *J. of Web Semantics*, 14:57–83, 2012.

[12] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, and G. Xiao. Ontop: Answering sparql queries over relational databases. *Semantic Web J.*, 8(3):471–487, 2017.

[13] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, and D. F. Savo. The Mastro system for ontology-based data access. *Semantic Web J.*, 2(1):43–53, 2011.

[14] P. P. Chen. The Entity-Relationship model: Toward a unified view of data. *ACM Trans. on Database Systems*, 1(1):9–36, Mar. 1976.

[15] B. Cuenca Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, and U. Sattler. OWL 2: The next step for OWL. *J. of Web Semantics*, 6(4):309–322, 2008.

[16] I. da Silva, G. Santucci, and C. del Sasso Freitas. Ontology visualization: One size does not fit all. In *Proc. of the 3rd Int. Eurovis Workshop on Visual Analytics (EuroVA)*, pages 91–95, 2012.

[17] K. X. de Souza, A. D. dos Santos, and S. R. Evangelista. Visualization of ontologies through hypertrees. In *Proc. of the 1st Latin American Conf. on Human-Computer Interaction*, pages 251–255. ACM, 2003.

[18] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

[19] A. Dix. *Human-computer interaction*. Springer, 2009.

[20] R. Falco, A. Gangemi, S. Peroni, D. Shotton, and F. Vitali. Modelling OWL ontologies with Graffoo. In *ESWC 2014 Satellite Events*, volume 8798 of *Lecture Notes in Computer Science*, pages 320–325, 2014.

[21] P. R. Fillottrani, E. Franconi, and S. Tessaris. The ICOM 3.0 intelligent conceptual modelling tool and methodology. *Semantic Web J.*, 3(3):293–306, 2012.

[22] M. R. Genesereth. Knowledge Interchange Format. In *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR)*, pages 599–600, 1991.

[23] J. H. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, N. F. Noy, and S. W. Tu. The evolution of Protégé: an environment for knowledge-based systems development. *Int. J. of Human-computer Studies*, 58(1):89–123, 2003.

[24] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang. Hermit: an OWL 2 reasoner. *J. of Automated Reasoning*, 53(3):245–269, 2014.

[25] B. C. Grau and B. Motik. Reasoning over ontologies with hidden content: The import-by-query approach. *CoRR*, abs/1401.5853, 2014.

[26] T. R. Gruber. Ontolingua: A mechanism to support portable ontologies. Technical report, Stanford University Knowledge Systems Laboratory, 1992.

[27] T. R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*. Kluwer Academic Publishers, 1993.

[28] N. Guarino, D. Oberle, and S. Staab. What is an ontology? In *Handbook on Ontologies*, pages 1–17. Springer, 2009.

[29] G. Guizzardi. *Ontological Foundations for Structural Conceptual Models*. PhD thesis, University of Twente, The Netherlands, 2005.

[30] P. Haase, H. Lewen, R. Studer, D. T. Tran, M. Erdmann, M. d'Aquin, and E. Motta. The NeOn ontology engineering toolkit. In *the 17th Int. World Wide Web Conf. – Developers Track*, 2008.

[31] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph. OWL 2 Web Ontology Language: Primer (second edition). W3C Recommendation, World Wide Web Consortium, Dec. 2012. Available at http://www.w3.org/TR/owl2-primer/.

[32] M. Horridge and P. F. Patel-Schneider. OWL 2 Web Ontology Language Manchester Syntax (second edition). W3C Working group note, World Wide Web Consortium, Dec. 2012. Available at https://www.w3.org/TR/owl2-manchester-syntax/.

[33] I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible $\mathcal{SROIQ}$. In *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR)*, pages 57–67, 2006.

[34] ISO/IEC, CH-1211 Geneva 20, Switzerland. *ISO/IEC 24707: Information technology–Common Logic (CL): A Framework for a Family of Logic-Based Languages*, 2nd edition, 2007.

[35] M. Kifer. Rule Interchange Format: The framework. In *Proc. of the 2nd Int. Conf. on Web Reasoning and Rule Systems (RR)*, volume 5341 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2008.

[36] M. Kifer, G. Lausen, and J. Wu. Logical foundations of Object-Oriented and frame-based languages. *J. of the ACM*, 42(4):741–843, 1995.

[37] S. Krivov, R. Williams, and F. Villa. GrOWL: A tool for visualization and editing of OWL ontologies. *J. of Web Semantics*, 5(2):54–57, 2007.

[38] D. Lembo, J. Mora, R. Rosati, D. F. Savo, and E. Thorstensen. Mapping analysis in ontology-based data access: Algorithms and complexity. In *Proc. of the 14th Int. Semantic Web Conf. (ISWC)*, volume 9366 of *Lecture Notes in Computer Science*, pages 217–234. Springer, 2015.

[39] D. Lembo, D. Pantaleone, V. Santarelli, and D. F. Savo. Easy OWL drawing with the Graphol visual ontology language. In *Proc. of the 15th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR)*, pages 573–576, 2016.

[40] D. Lembo, D. Pantaleone, V. Santarelli, and D. F. Savo. Eddy: A graphical editor for OWL 2 ontologies. In *Proc. of the 25th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 4252–4253, 2016.

[41] D. Lembo, R. Rosati, M. Ruzzi, D. F. Savo, and E. Tocci. Visualization and management of mappings in ontology-based data access (progress report). In *Proc. of the 27th Int. Workshop on Description Logic (DL)*, volume 1193 of *CEUR Electronic Workshop Proceedings,* http://ceur-ws.org/, pages 595–607, 2014.

[42] D. Lembo, V. Santarelli, and D. F. Savo. The Graphol language for OWL 2 ontology editing and visualization, 2017. Manuscript. Available at http://obdasystems.com/sites/default/files/graphol.pdf.

[43] D. B. Lenat and R. V. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Addison Wesley Publ. Co., 1990.

[44] M. Lenzerini. Ontology-based data management. In *Proc. of the 20th Int. Conf. on Information and Knowledge Management (CIKM)*, pages 5–6, 2011.

[45] R. Liepins, M. Grasmanis, and U. Bojars. OWLGrEd ontology visualizer. In *Proc. of ISWC Developers Workshop*, volume 1268 of *CEUR Electronic Workshop Proceedings,* http://ceur-ws.org/, pages 37–42, 2014.

[46] S. Lohmann, S. Negru, F. Haag, and T. Ertl. Visualizing ontologies with VOWL. *Semantic Web J.*, 7(4):399–419, 2015.

[47] B. Motik, B. Parsia, and P. F. Patel-Schneider. OWL 2 Web Ontology Language structural specification and functional-style syntax (second edition). W3C Recommendation, World Wide Web Consortium, Dec. 2012. Available at http://www.w3.org/TR/owl2-syntax/.

[48] B. Motik, B. Parsia, and P. F. Patel-Schneider. OWL 2 Web Ontology Language XML Serialization (second edition). W3C Recommendation, World Wide Web Consortium, Dec. 2012. Available at https://www.w3.org/TR/2012/REC-owl2-xml-serialization-20121211/.

[49] E. Motta. An overview of the OCML modelling language. In *Proc. of the 8th Workshop on Knowledge Engineering Methods and Languages (KEML)*, 1998.

[50] R. D. Murray. Python 3.4. Python Documentation, Mar. 2014. Available at https://docs.python.org/3.4/whatsnew/3.4.html.

[51] J. Nielsen. *Usability engineering*. Academic Press, 1993.

[52] J. Nielsen. Enhancing the explanatory power of usability heuristics. In *In Proc. of the ACM Conference on Human Factors in Computing Systems (CHI)*, pages 152–158. ACM Press, 1994.

[53] A. A. Romero, M. Kaminski, B. C. Grau, and I. Horrocks. Module extraction in expressive ontology languages via datalog reasoning. *J. of Artificial Intelligence Research*, 55:499–564, 2016.

[54] D. F. Savo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodríguez-Muro, V. Romagnoli, M. Ruzzi, and G. Stella. MASTRO at work: Experiences on ontology-based data access. In *Proc. of the 23rd Int. Workshop on Description Logic (DL)*, volume 573 of *CEUR Electronic Workshop Proceedings,* http://ceur-ws.org/, pages 20–31, 2010.

[55] N. Shadbolt, W. Hall, and T. Berners-Lee. The Semantic Web revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006.

[56] B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Trans. on Graphics*, 11(1):92–99, 1992.

[57] B. Shneiderman. *Designing the user interface: strategies for effective human-computer interaction*. Pearson Education India, 2010.

[58] S. Staab and R. Studer, editors. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, 2nd edition, 2009.

[59] G. Stapleton, J. Howse, K. Taylor, A. Delaney, J. Burton, and P. Chapman. Towards diagrammatic ontology patterns. In *Proc. of the 4th Workshop on Ontology and Semantic Web Patterns (WOP)*, 2013.

[60] R. Studer, V. R. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *Data and Knowledge Engineering*, 25(1-2):161–197, 1998.

[61] Unified Modeling Language (UML) superstructure, version 2.0. Available at http://www.uml.org/, Aug. 2005.

[62] M. Q. Wang Baldonado, A. Woodruff, and A. Kuchinsky. Guidelines for using multiple views in information visualization. In *Proc. of the 1st Working Conf. on Advanced Visual Interfaces (AVI)*, pages 110–119. ACM, 2000.

[63] M. Weiten. Ontostudio® as a ontology engineering environment. In *Semantic Knowledge Management*, pages 51–60. Springer, 2009.