Aided OWL Notation (AOWLN): Conceptual Modelling and Visualisation of Advanced SWRL Rules

Johannes Nguyen, Jannik Geyer, Thomas Farrenkopf and Michael Guckert
KITE - Technische Hochschule Mittelhessen, Wilhelm-Leuschner-Straße 13 , Friedberg, Germany

Keywords: Graphical Rule Representation, SWRL, OWL, Rule Visualisation.

Abstract: Ontologies are a common and generally accepted instrument for the documentation of knowledge in a formalised machine readable form. This paper focuses on ontologies encoded in *Web Ontology Language (OWL)*. OWL is description-logic based and can be extended by using *Semantic Web Rule Language (SWRL)* to express Horn clause like rules that allow the ontology to go beyond the scope of the more object-centric description logic propositions. The combination of OWL and SWRL has proved to be highly useful in practical applications. However, SWRL rules soon become complex and confusing in mere textual representations. This particular issue becomes obvious when ontologies grow in size and the number of rules increases. A solution for this problem can be an appropriate graphical representation of the rules. This paper proposes a graphical visualisation concept for SWRL rules that we call *Aided OWL Notation (AOWLN)*. Additionally, we present a prototypical *Protégé* plugin that automatically visualises rules.

1 INTRODUCTION

Gruber described ontologies to be an *explicit specification of a conceptualisation* which was later reformulated by Borst to also be *shared* (see (Guarino et al., 2009) for a discussion). Ontologies define concepts together with their properties and interrelations in a formal and machine readable form. They have become a common instrument for consistently documenting knowledge to be shared among humans and machines. Having the idea of supporting interoperability as a major purpose, standardisation is an important issue. W3C has standardised *OWL*. Being an object-centric language OWL has limitations in express rather simple if-then constructions, e.g. for asserting data or object properties. Therefore, the combination of *OWL* together with *Semantic Web Rule Language (SWRL)* is a frequently used set of tools.

1.1 Motivation

SWRL extends *OWL* with the ability to use Horn clause like if-then rules. *SWRL* is a proposed W3C standard that has gained popularity in recent years. With a growing developer base, the lack of options to visualise rules becomes more and more relevant. In February 2017 Martin O'Connor (creator of Protégé) and other developers addressed this topic in a web forum, resulting in an open call for an appropriate visualisation tool.[1] Although this is not a new debate, most existing solutions have limitations, making the development of an easy to use visualisation tool for SWRL a necessity.

1.2 Problem

Rules expressed with SWRL increase the visual complexity of an ontology. With complex mathematical formulas invoking SWRL built-ins, rules can easily stretch over several lines which makes reading and comprehending difficult. However, in practical industrial applications highly complicated rules cannot be avoided as expressing engineering problems requires mathematical formulas that lead to extensive use of built-ins inducing high complexity. This makes it difficult if not impossible to follow the logical structure of the rules, even for domain experts. So far, Protégé only offers limited options to search and identify specific rules. An integrated graphical visualisation will lead to higher transparency and a better overview, making the rules more comprehensible. A new graphical notation format for SWRL rules with higher transparency needs to be defined.

[1] See http://protege-project.136.n4.nabble.com/Visualisation-tool-for-OWL-and-SWRL-rules-td4667578.html - (accessed on 06/01/2018)

Nguyen, J., Geyer, J., Farrenkopf, T. and Guckert, M.
Aided OWL Notation (AOWLN): Conceptual Modelling and Visualisation of Advanced SWRL Rules.
DOI: 10.5220/0006917701750182
In *Proceedings of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2018) - Volume 2: KEOD*, pages 175-182
ISBN: 978-989-758-330-8

1.3 Idea

The contribution of this work is a proposal for a set of notation elements that are able to portray the main logical constructs used in SWRL. We will specify an algorithm that composes the newly defined notation elements into a diagram. In our initial approaches we attempted to model SWRL rules using UML and BPMN notation elements. However, the elements available there are semantically inappropriate and can lead to BPMN and UML diagrams contradicting the SWRL logic. Furthermore, we present prototypical visualisation tool that can be implemented as a Protégé plugin.

1.4 Outline of the Paper

The following section provides a technology review, giving an overview of the most important existing solutions for visualising SWRL rules, inter alia, *Protégé Axiomé* which is a known visualisation plugin for Protégé. Section 3 introduces a new set of notation elements for the graphical visualisation and in section 4 an example is presented as proof of concept. This example demonstrates the transformation of rule given in textual description into our graphical notation. Finally, the implementation of the visualisation plugin is described.

2 RELATED WORK

In this section, other approaches to visualise rules in ontologies are discussed.

2.1 Protégé Axiomé

Protégé Axiomé was developed by the creators of Protégé. It is a plugin that consists of five main functional areas (Hassanpour et al., 2009). The first functional area is a rule graph for visualising the interdependencies between individual rules to represent an entire rule base. The rules are portrayed as nodes that are connected through directed edges. Besides this, the tool also offers an option for rule paraphrasing. This option enables the creation of English like text representations for individual rules and categorises them based on their syntactic structure. Additionally, a rule elicitation function was implemented that creates templates for adding new rules to the rule base. Protégé Axiomé visualisation uses a tree structure created by applying *Depth First Search* to chain the variables (Hassanpour et al., 2009). By chaining variables a simple flowchart diagram which visualises the basic logical sequence for data and object properties is derived. However, the tool neither differentiates between existing SWRL elements nor does it clarify the application of built-ins (see Fig. 1). A more appropriate modelling concept for visualisation should include more information about the rule and its constituents. Furthermore, the tool at this stage is not up-to-date as it has not been ported to the current version of Protégé 5.0.

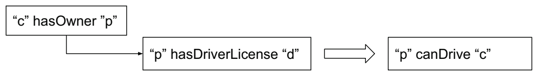


Figure 1: Axiomé – individual rule visualisation.

2.2 Poznan University of Technology Graph-based Editor

In 2013, *Jaroslaw Bak* and his team from *Poznan University of Technology* published a prototypical implementation of a graphed-based editor for SWRL rules. The editor focuses on the visualisation of individual SWRL rules. It is based on a set of new modelling elements for the illustration of rules in diagrams (see Fig. 2).

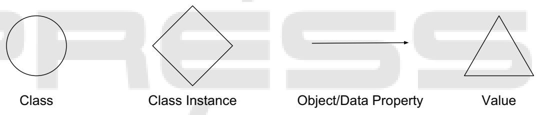


Figure 2: Set of notation elements.

This approach distinguishes separate taxonomies for *Classes, Data Properties,* and *Object Properties* (Bak et al., 2013). The editor creates a diagram for each of the three taxonomies, allowing the user to choose between different perspectives. The diagrams are split into two sections (see Fig. 3) (Bak et al., 2013). The left-hand section portrays the condition elements (antecedent) of an SWRL rule whereas the right-hand section shows the conclusion (consequent).

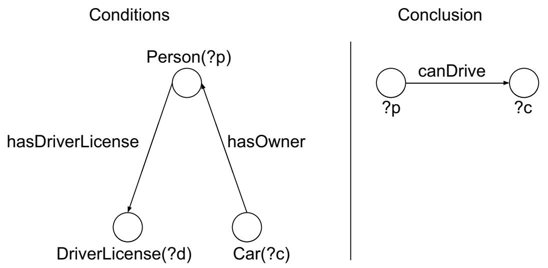


Figure 3: Poznan Graph Editor - SWRL visualisation.

In general, this modelling approach offers a lot of potential as it is visually self-explaining, but it is limited when trying to show the impact of built-ins or restrictions. However, built-ins are essential components of SWRL necessary to implement mathematical operations. The separation of the representation into different views can be considered a deficiency. A unified diagram for all three perspectives may improve general understanding of the rule. Despite this, we agree that the visual separation of rule antecedent and consequent is as a good approach.

2.3 Graph Inscribed Logic (Grailog)

Grailog is a combination of generalised graph constructs for visualising data, inter alia, *Horn logic* (Boley, 2013). The visualisation is adapted to the industry standard *RuleML*. With Grailog a new concept of *hypergraphs* is introduced which is a proposed enhancement in comparison to directed labelled graphs (Boley, 2013). According to Boley, directed labelled graphs (DLG) are a good starting point for visual knowledge representation but come with major disadvantages when trying to illustrate non-binary relationships (Boley, 2013). This is why he created *hyperarcs* as specialised arrows for his notation. The following graphs show a comparison of a hypergraph on the left-hand side and a directed labelled graph on the right-hand side (see Fig. 4).

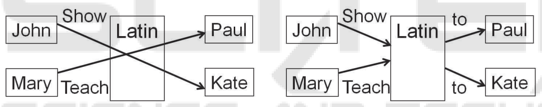


Figure 4: Hypergraphs in comparison to DLG 1 (Boley, 2013).

Both diagrams in Fig. 4 show the two statements:

"John shows Latin to Kate"
"Mary teaches Latin to Paul"

Using a directed labelled graph has the disadvantage of losing the context of input and output arrows (Boley, 2013). This means that the graph may also be misinterpreted as *"John shows Latin to Paul"* and *"Mary teaches Latin to Kate"*. However, the hyperarcs provide a means to unmistakably define the non-binary relationship. When trying to correctly and unambiguously illustrate the two given sentences as a DLG, the graph becomes significantly more complex, demonstrating a major advantage of Grailog Hypergraphs (see Fig. 5).

Grailog also offers the possibility to formulate advanced logic using the idea of so called *complex nodes*. According to Grailog, a graph can consist of *elementary nodes* such as *John* and *Kate*. Moreover, a complex node is able to contain other graphs, making it an

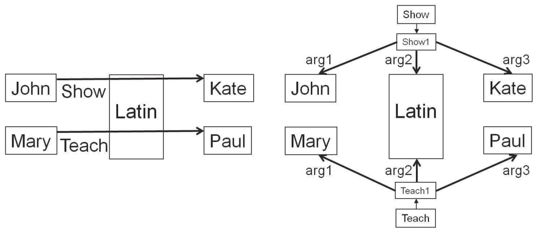


Figure 5: Hypergraphs in comparison to DLG 2 (Boley, 2013).

enclosing entity. Based on this, it is also possible to express Horn Logic using a combination of complex nodes in Grailog (see Fig. 6) (Boley, 2013). Although Grailog can describe Horn Logic, it is not specialised for SWRL making it difficult to portray SWRL built-ins. In this paper, we focus on a more specialised solution for SWRL rules.

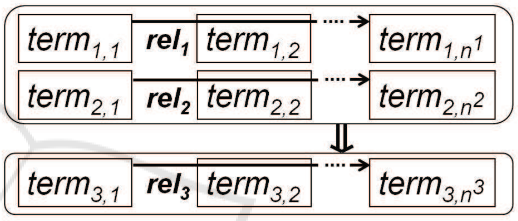


Figure 6: Grailog - Horn logic (Boley, 2013).

2.4 Using UML State Diagrams for Visualising Business Rules

In 2008, *Konrad Kułakowski* and *Grzegorz J. Nalepa* published a modelling approach for business rules using UML diagrams. The main idea is to represent a rule base as a class diagram. Based on this, each class represents a single rule. The class diagram then shows dependencies between different rules (Kulakowski and Nalepa, 2008). Furthermore, each class has its own state diagram which is described as a *rule definition diagram*. The paper defines rules as plain textual if-then statements (Kulakowski and Nalepa, 2008). The rule definition diagram expresses conditions using the UML standard *Object Constraint Language (OCL)*. As the proposed modelling format is designed for business rules in general, the possibilities to visualise more complex SWRL logic rules are limited.

3 CONCEPTUAL MODELLING

Before we define our notation elements we give a brief discussion of relevant aspects of ontologies and rules.

3.1 OWL/SWRL Elements

OWL is a standardised general knowledge representation language for creating ontologies (McGuinness et al., 2004). The elements of OWL are often denominated as *atoms* when used in rules.
An ontology O is a triple ($\mathcal{C}$,$\mathcal{R}$,I). $\mathcal{C}$ is a set of concepts, $\mathcal{R}$ a set of relations, and I a set of individual objects. Concepts (i.e. Classes) formally denote sets of objects. From a perspective of formal logic, these sets are the extension of concepts while concepts define the intentional representation of the corresponding set of objects. An object that belongs to a concept is called an instance of that concept. The elements of $\mathcal{R}$ are relations (also called roles or object properties, as they manifest links on the level of the instances) having subsets of $\mathcal{C}$ as domain and range. The extension of a role then is a set of pairs (c, d) with c, d elements of I. Additionally, instances can have data properties through which they get linked to primitive data e.g. strings or numbers. Typically, ontologies are formulated by means of description logics with differing levels of expressiveness (Baader, 2003). Usually description logics are proper subsets of first order logic where expressiveness has been traded for decidability. Inference knowledge is implicitly given by the underlying mechanisms of the available reasoning instruments. Apart from this, SWRL as an extension also includes elements for *restrictions* and *built-ins* (McGuinness et al., 2004). Built-ins can be described as operators similar to methods or functions respectively in conventional programming languages. For instance, mathematical operations and data type restrictions can be expressed through built-ins. The following table summarises the most significant elements and the corresponding syntactic structure when used in SWRL rules.

Table 1: OWL/SWRL Elements.

OWL/SWRL Element	**Syntax**
Class	Person
Individual / Class instance	Person(bob) / Person(alice)
Data Property	canDrive(bob, true)
Object Property	isSon(bob, alice)
Built-ins	swrlb:lessThan(?*age*,18)
Restrictions	integer[$>$ 0]
Variable	?*age*

3.2 SWRL Syntax Diagram

Horrocks (2004) defines a SWRL rule as a pair *(A, C)* in which both, *antecedent* **A** and *consequent* **C** are sets of *atoms* $\mathbf{S_A}$ and $\mathbf{S_C}$ (see Fig. 7).

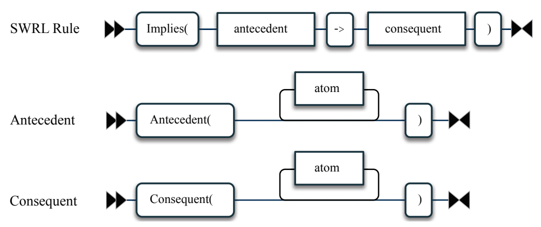


Figure 7: SWRL syntax diagram 1.

An atom element r_A must be declared with an *atom name* $\in B_N \cup C_N \cup O_P \cup D_P \cup D_R$ for which B_N is the set of *built-in names*, C_N the set of *class names*, O_P the set of *object property names*, D_P the set of *data property names* and finally D_R the set of *data ranges*. A built-in name b_N is followed by zero or one *built-inID* and at least one or more *d-objects*. The built-in ID is a variable in which the built-in output is written. A d-object can be a *data literal* or a *d-variable* which in the second case is an *URI reference* to an entity defined in the ontology. A class name C_N is followed by an *i-object* which can either be an individual-ID or an *i-variable*. An i-variable is an URI reference to an entity defined in the ontology. Apart from this, an object property name o_P is followed by a pair *(i-object, i-object)* and a data property name d_P complemented by a pair *(i-object, d-object)*. A data range d_R is followed by a single d-object (see Fig. 8).

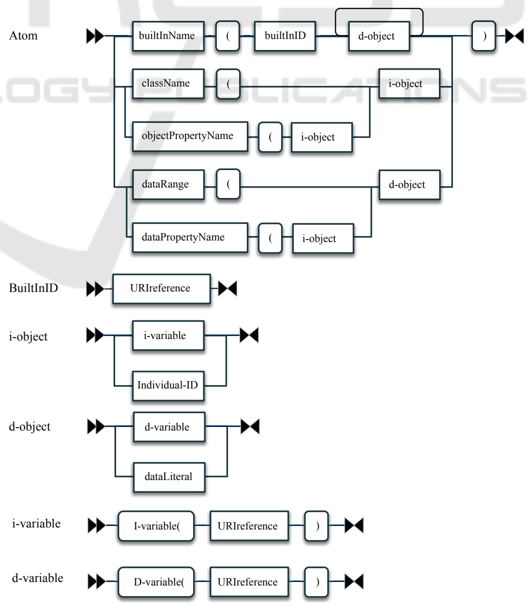


Figure 8: SWRL syntax diagram 2.

3.3 AOWLN Graph Structure

As discussed before, directed labelled graphs are well suited to visualise logical sequences (see Section 2). In order to visualise SWRL rules, a corresponding graphical symbol for each of the existing semantic OWL/SWRL elements has been mapped and can be composed into a graph.

For a given SWRL rule R_A we define a *Graph G* and map the components of R_A to nodes of G. The graph consists of a heterogeneous set of vertices $V = R \cup O \cup T \cup D$ (R being a set of *rectangles* (class nodes), O a set of *ovals* (property nodes), T a set of *trapeziums* (variable nodes) and D a set of *diamond shapes* (built-in collection nodes) and edges $E = D_E \cup O_E \cup B_E$) ($D_E$ being a set of *unlabelled solid arrows* (data property edges), O_E a set of *dashed unlabelled arrows* (object property edges) and B_E a set of *solid and labelled arrows* (built-in edges). The mapping is described in figure 9.

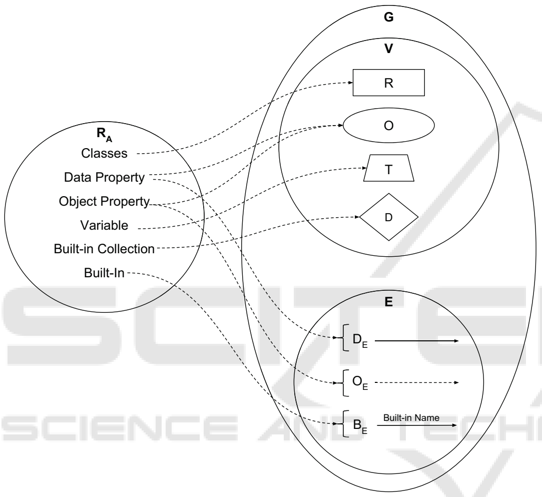


Figure 9: AOWLN elements.

According to this mapping, classes are represented by rectangle shapes and properties by oval shapes. Properties associated with classes are connected to the corresponding class element with an edge. Edges are unlabelled and directed. While class nodes and data properties as well as variables are connected with solid lines, edges associated with object properties are dashed. Variables are represented by trapezium shapes. In SWRL data type restrictions and operations can be expressed with built-ins. Data type restrictions are written into the lower half of a property symbol, separated by a dashed line. Moreover, built-ins are visualised by labelling the edge that progresses towards the resulting variable. If multiple built-ins result in the same variable, these built-ins are summarised into a new diamond shaped *Built-in Collection* element.

For both rule antecedent and consequent separate graphs are created which are displayed in juxtaposition. The following pseudocode sketches the creation of the graph.

```
List aowlnElements
List aowlnEdges

function createAOWLNGraph(RuleFragment rf)
 for each classAtom in rf
  if classAtom not in aowlnElements
   aowlnElements add classAtom
   completeLogicalSequence(classAtom, rf)

   for each ObjectPropAtom in rf
    if classAtom is firstArg:
     appendAtom (classAtom, ObjectPropAtom)
     classAtom = ObjectPropAtom.SecondArg
     completeLogicalSequence(classAtom, rf)
    endIf
  endIf
 endFor
 aowlnGraph = new Graph(aowlnElements, aowlnEdges)

function appendAtom (startAtom, nextAtom)
 aowlnElements add nextAtom
 aowlnEdges  add new Edge(startAtom, nextAtom)

function completeLogicalSequence(classAtom, ruleFragment)
 for each DataPropAtom in ruleFragment
  if classAtom is firstArg:
   appendAtom (classAtom, DataPropAtom)
   currentAtom = DataPropAtom
   while (successorAtom not null) do
    successorAtom =
     findLogicalSuccessor(successorAtom, ruleFragment)
    appendAtom(currentAtom, successorAtom)
    currentAtom = successorAtom
   endWhile

   varElement = currentAtom.getResultingVar()
   appendAtom(currentAtom, varElement)
  endIf
 endFor
```

By convention, for each variable instance of a class, a class node is created explicitly. Furthermore, the last variable of a logical sequence must be indicated. A logical sequence is the concatenation of SWRL atoms through joint references to variables. However, variables that are passed on to consequential properties or built-ins may be skipped in the visualisation to avoid redundant graph elements. For logical sequences that end with a class atom node (i.e. object properties) the variable node at the end of the sequence can be omitted as class atoms are explicitly created for each variable instance of a class.

The algorithm described above requires a sequential ordering of the atoms in the SWRL rule:

1. Class 1
   a. class1DataProperty1
   b. class1**T**ransitive**N**ext**P**roperty1
      i. c1TNP1BuiltIn1
      ii. c1TNP1BuiltIn2
      iii. c1TNP1Restriction
   c. class1TNP2

   d. class1TNPN
2. Class2
3. ...
4. ClassN

(1) objectProperty1
(2) objectProperty2
(3) ...
(4) objectPropertyN

⇒ Conclusion

The rule conclusion is processed just like that. The ordering ensures the affiliation of transitive elements simplifying the implementation. Built-ins which only use a single variable in position one will not be visualised. See the following example:

1. **add(?ageNew, ?age, 10)**
2. **add(?ageNew, 6, 10)**

The SWRL math built-in, *swrlb:add()* is used. This built-in is satisfied if the first argument is equal to the arithmetic sum of the following arguments (Horrocks et al., 2004). Hence, the built-in writes the result into the first variable *?newAge*. In the first case, the variable *?newAge* is derived from the variable *?age*. In the second case the operands of the summation are constants and will not be visualised because they are not part of a logical sequence. This enhances the readability of the graph by eliminating nodes of low meaningfulness.

4 EXAMPLE OF APPLICATION

The following example demonstrates the use of the notation elements.

Textual Description:
Individuals under the age of 18 as a potential driver of a vehicle with a weight of less than 26,000 lbs are able to drive in California if they possess an out-of-state driver's license except for New York and are visiting the state for less than 10 days. For this, the person must be the owner of the car.

Table 2: AOWLN Elements Mapping.

Concept / Class	**Data Property** → **Built-In**	**Object Property**
Person(?p)	hasAge(?p,?g) → lessThan(18)	ownsCar(?p,?c)
Car(?c)	numberOfVisiting-InDaysInCA(?p,?x) → lessThan(10)	hasDriverLicence(?p,?d)
DriverLicense(?d)	issuedStateOf(?d,?s) → notEqual(s,"CA") → notEqual(s,"NY")	
	hasWeightInLBS(?x,?w)	

SWRL:
$Person(?p) \wedge hasAge(?p,?g) \wedge swrlb: lessThan(?g,18) \wedge numberOfVisitngDaysInCA(?p,?x) \wedge swrlb: lessThan(?x,10) \wedge DriverLicense(?d) \wedge hasDriverLicense(?p,?d) \wedge issuedInStateOf(?d,?s) \wedge swrlb: notEqual(?s,''CA'') \wedge swrlb: notEqual(?s,''NY'') \wedge Car(?c) \wedge hasWeightInLBS(?c,?w) \wedge swrlb: lessThan(?w,26000) \wedge ownsCar(?p,?c) \Rightarrow Person(?p) \wedge Car(?c) \wedge canDrive(?p,?c)$

AOWLN:
As the Fig. 10 displays, there is a separation of rule conditions and the rule conclusion similar to the Poznan graph editor (see Section 2.2). This separation helps making the graph more manageable when rules are getting more complex. However, the visualisation does not use separate taxonomies. Built-ins are included in the visualisation. The names of the built-ins are used to label the edges. In case that multiple built-ins result in the same variable, the built-ins are summarised in a built-in collection symbol. Fig. 10 shows, that it is possible to reduce the rule illustration to the most essential and necessary elements to portray the logical sequence. This makes working with rule based systems significantly easier.

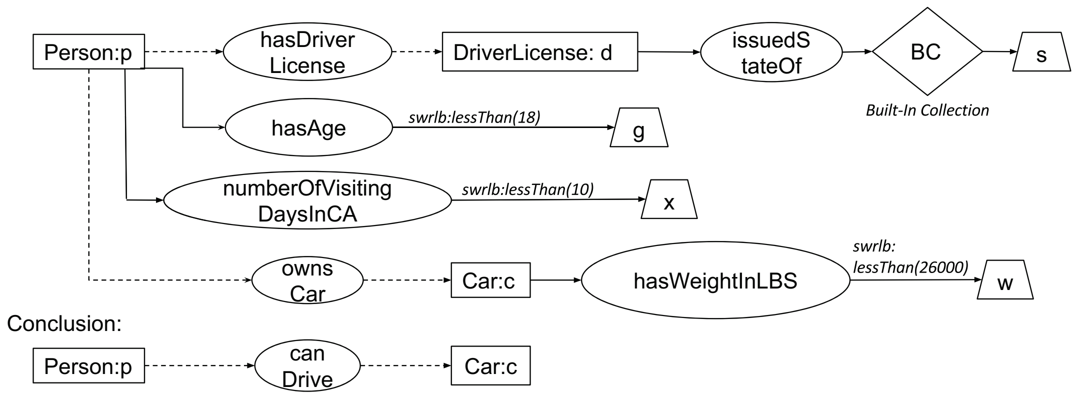


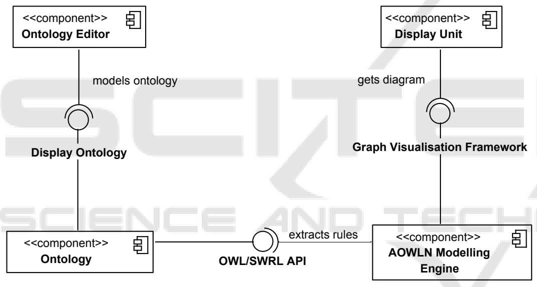
Figure 10: Example rule – AOWLN graph.

5 IMPLEMENTATION

In this section we address the prototypical implementation of the visualisation plugin for Protégé.

5.1 Design

The plugin will be seamlessly integrated into the Protégé environment. The application includes a *full-text search* to simplify the work with the rule set. Based on this, the user is given the option to select rules for which an AOWLN diagram will be created. The implementation of the visualisation engine is separated from the display unit and the ontology. This makes the engine independent for future applications as shown in the following component diagram. A completely assembled system can use Protégé as an ontology editor to document and generate the ontology. The AOWLN modelling engine extracts SWRL rules through the OWL/SWRL API and produces a diagram as an image file or as an interactive graph (see Fig. 11).



<<component>> Ontology Editor
<<component>> Display Unit
models ontology
gets diagram
Display Ontology
Graph Visualisation Framework
<<component>> Ontology
extracts rules
OWL/SWRL API
<<component>> AOWLN Modelling Engine

Figure 11: Component diagram.

5.2 Modelling Engine for Graphical Abstraction (MEGA)

The implementation is based on a three-layer-approach has been developed to process rule data from the ontology into the required graph format that can be visualised (see Fig. 12). In *Layer 0*, the *SWRL API*[2] is the fundamental component. The SWRL API (an extension of the OWL API) reads the ontology and provides a set of SWRLAtoms. As the SWRLAtom class from the SWRL API includes unnecessary information for the visualisation, a custom SWRL atom structure has been defined which is illustrated as *Layer 1*. The *createCustomSWRLAtoms()* method transforms the given SWRL atoms from the SWRL API into the custom SWRL atom structure. This makes it easier to focus on the most essential information for the visualisation. In a second step, the *createAOWLNElements()* method creates a separate list for both, AOWLN nodes and edges for *Layer 2* based on the custom SWRL atoms in *Layer 1*. Edges contain references for a start and a target node.

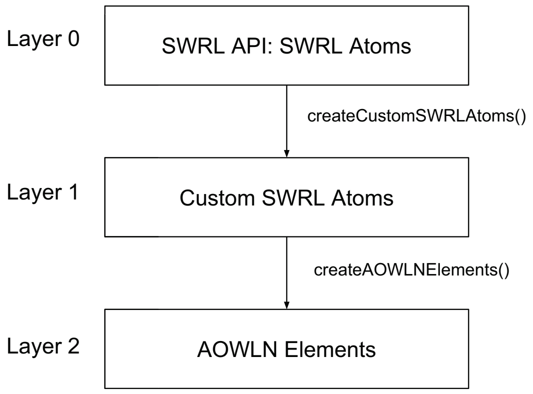


Figure 12: MEGA – three-layer-approach.

5.3 Prototype

The prototype has the following features: *search and select rules* from the rule base, *visualise the rules* as an AOWLN graph and it includes an option to *edit and update* a selected rule in the rule base. The prototypical Protégé plugin is written in Java as both, the Protégé environment and the SWRL API are also implemented in Java. For the graph visualisation, we make use of the open source library *Graphviz*.[3]

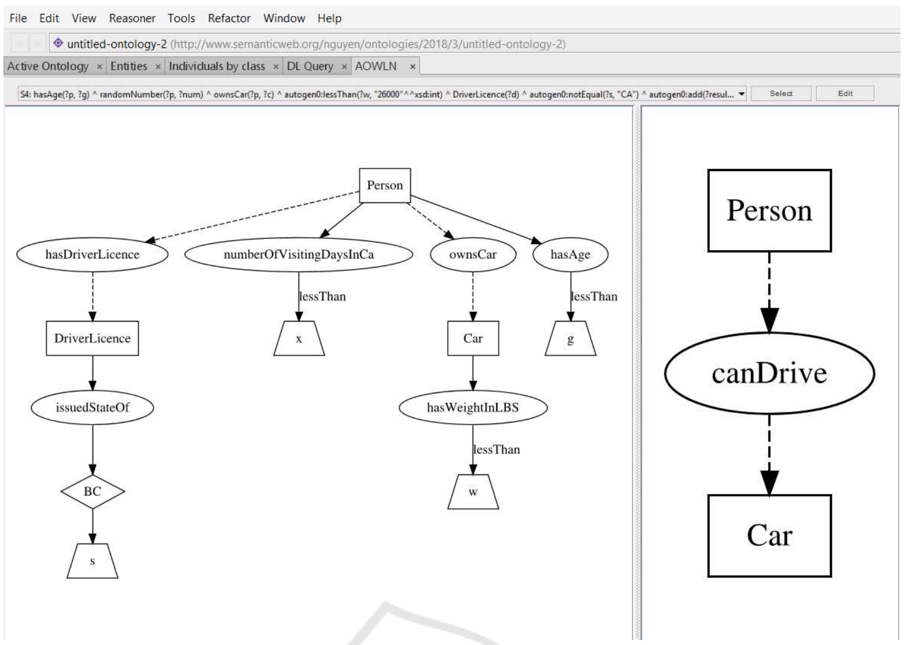
6 CONCLUSION AND FUTURE WORK

This paper defines a graphical notation format for the visualisation of SWRL rules and describes a prototypical Protégé plugin for rule visualisation. AOWLN offers means for visualising all types of components of SWRL elements especially built-ins. For this, the form of representation is a pair of directed labelled graphs to portray rule antecedent and consequent. The implemented plugin offers options to *search and select rules* from the rule base, *visualise the rules* as an AOWLN graph and includes an option to *edit and update* a selected rule. As proof of concept the plugin is currently used in an industrial digitisation project which enabled several feedback loops.

Further functions to improve usability and error handling will make working with the visualisation tool

[2]See https://github.com/protegeproject/swrlapi - (accessed on 06/01/2018)

[3]See https://www.graphviz.org/ - (accessed on 20/01/2018)



```
S4: hasAge(?p, ?g) ^ randomNumber(?p, ?num) ^ ownsCar(?p, ?c) ^ autogen0:lessThan(?w, "26000"^^xsd:int) ^ DriverLicence(?d) ^ autogen0:notEqual(?s, "CA") ^ autogen0:add(?resul...
```

Figure 13: AOWLN Protégé Plugin – Protoype.

more efficient. Since the prototype was written as a component independent from both the ontology and the display unit, it can easily be integrated into larger systems e.g. the plugin can easily be extended to a graphical editor that allows for interactive creation and modification of SWRL rules. Further formalisation and generalisation to make the algorithm work with arbitrary SWRL rules will be carried out. We will provide the source code for the AOWLN project on github. The repository can be accessed under the following URL: https://github.com/KITE-Cloud/AOWLN

REFERENCES

Baader, F. (2003). *The description logic handbook: Theory, implementation and applications*. Cambridge university press.

Bak, J., Nowak, M., and Jedrzejek, C. (2013). Graph-based editor for swrl rule bases. In *RuleML (2)*. Citeseer.

Boley, H. (2013). Grailog 1.0: Graph-Logic Visualization of Ontologies and Rules. In *Proc. 7th International Web Rule Symposium: Research Based and Industry Focused (RuleML 2013), Seattle, Washington, USA*, volume 8035 of *Lecture Notes in Computer Science*, pages 52–67. Springer.

Guarino, N., Oberle, D., and Staab, S. (2009). What is an ontology? In *Handbook on ontologies*, pages 1–17. Springer.

Hassanpour, S., O'Connor, M. J., and Das, A. K. (2009). Axiomé: A tool for the elicitation and management of SWRL rules. In *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2009), Chantilly, VA, United States, October 23-24, 2009*.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., Dean, M., et al. (2004). Swrl: A semantic web rule language combining owl and ruleml. *W3C Member submission*, 21:79.

Kulakowski, K. and Nalepa, G. J. (2008). Using uml state diagrams for visual modeling of business rules. In *Computer Science and Information Technology, 2008. IMCSIT 2008. International Multiconference on*, pages 189–194. IEEE.

McGuinness, D. L., Van Harmelen, F., et al. (2004). Owl web ontology language overview. *W3C recommendation*, 10(10):2004.