Semantic Web 0 (0) 1
IOS Press


# A Visual Modeling Approach for the Semantic Web Rule Language

Benedikt Pittl [a], Hans-Georg Fill [a,b,*]
[a] *Research Group Knowledge Engineering, University of Vienna, Austria*
*E-mail: benedikt.pittl@univie.ac.at*
[b] *System Development and Database Application Group, University of Bamberg, Germany*
*E-mail: hans-georg.fill@univie.ac.at*

**Abstract.** The Semantic Web Rule Language (SWRL) is considered a main pillar for realizing the semantic web and for supporting innovative rule-based applications. Thereby, it is used to infer new knowledge from a given fact base. Today, SWRL rules are developed and managed by technical experts in text-based editors using software applications such as the Stanford Protégé toolkit. For easing the specification and analysis of SWRL rules by non-technical users, we introduce in this paper a visual modeling approach for SWRL. By building upon a visual modeling language, the approach includes validation mechanisms and layouting algorithms for visually representing new as well as existing rules. The approach further provides import and export interfaces to common SWRL exchange formats. In this way, its compatibility with widely-used reasoners and semantic web platforms is guaranteed. For ensuring its feasibility, the approach has been prototypically realized using the SeMFIS platform and evaluated using the sample rules as provided in the SWRL specification.

Keywords: SWRL, OWL, Modeling Language, Semantic Processing, SeMFIS

## 1. Introduction

Rule-based systems as well as the corresponding formalisms constituted for a long time an important research area for expert systems [1]. Today, they have become popular again, especially for semantic web-based applications and for enterprise information systems [2]. For example, in the domain of enterprise information systems, rules are today being applied for specifying business process decisions and constraints as well as for compliance checking, e.g. [3, 4]. The Semantic Web Rule Language (SWRL) is thereby considered as a significant technology towards the semantic web that is adequate for many types of information systems [5]. In particular, SWRL is a sound and well-established technique for deriving new facts from a given fact base. Hence, the need for comprehensive SWRL editors is obvious and explicitly mentioned in the scientific community, e.g. [6–8]. Today, SWRL rules are usually developed and modified using text-based editors such as the Protégé SWRL tab [5]. The text-based representation is convenient for technical experts but comes in a mathematical notation, which is not common to many domain experts. The representation thus obviously impedes a more wide-spread adoption of SWRL. Recently, Skillen et al. showed empirically that business users and even technical experts would benefit from a visual modeling language for SWRL in terms of understandability and development time [8]. The subsequently developed visual modeling language shows a strong focus on usability but does not support all SWRL concepts [8]. Further, the import function as well as the handling of the underlying OWL ontology is limited - see section 2 for an in-depth discussion.

In a previous paper we therefore proposed a first idea for a visual modeling language that covers all concepts of SWRL [9]. The paper at hand extends this idea and contains an in-depth description of a comprehensive, visual modeling approach for SWRL. The main goal is enable the visual modeling of SWRL that can

*Corresponding author. E-mail: hans-georg.fill@univie.ac.at.

be used both for use cases dealing with the analysis of existing SWRL rules, e.g. to enhance the understanding of a rule, as well as for use cases where SWRL rules have to be specified by non-technical experts, e.g. to conduct compliance checking by business users [4]. A further goal is to realize a modeling language that includes all necessary constructs in a visual form and that is thus easy to understand even by non-technical users. In more detail, this led us to the following list of requirements for such a modeling approach:

(i) Visual modeling of all atoms as defined in the SWRL specification (ii) Explicit visual representation of variables and data values (iii) Linkage to visually represented OWL ontologies using the previously developed SeMFIS modeling language [10, 11] (iv) Serialization and De-serialization of visual SWRL models in OWL-XML syntax (v) Prototypical implementation of the approach to evaluate its feasibility.

In this paper we describe how we met these requirements. In particular, the contributions are as follows:

- Detailed description of the metamodel of the visual modeling language
- Formalization of the modeling language for an implementation-independent representation
- Development of implementation-independent rules for transforming the model to concrete syntax representations and vice versa
- Description of the prototypical implementation using the SeMFIS platform based on ADOxx [12]

The remainder of the paper is structured as follows: In section 2 foundations of conceptual modeling and SWRL as well as related work is analyzed. The modeling language for SWRL is introduced in section 3. In section 4 the transformation from the SWRL model to a SWRL syntax and vice versa is introduced. The technical implementation of the approach is introduced in section 5 followed by a use case which is presented in section 6. In section 7 the limitations of the approach are discussed. The paper ends with a conclusion in section 8.

## 2. Foundations and Related Work

In this section we discuss at first foundations of conceptual modeling that are necessary for describing our approach. This includes a characterization of modeling methods as it is typically used in the area of enterprise modeling and the formalization of modeling languages. Second, we regard the contents of the W3C SWRL proposal that is the basis for the subsequently presented modeling language.

### 2.1. Foundations of Conceptual Modeling Methods

According to Karagiannis and Kühn, a modeling method consists of (i) a modeling language, which encompasses a visual notation, syntax and semantics, (ii) a modeling procedure, which describes how to use a modeling language, and (iii) algorithms and mechanisms, which can be applied to the models [13]. In the paper at hand we focus on the description of the modeling language as well as on algorithms and mechanisms.

For describing modeling languages, specific formalisms have been designed in the past [14, 15]. Examples include EMF [16] or a specific formalism for the UML OCL specification[1]. Four our purposes, we decided to use the ***F**ormalism for **D**escribing ADOxx **M**eta Models and **M**odels* (FDMM) [17]. It has been used for formalizing a wide variety of modeling languages in the past, e.g. [18–20]. FDMM supports the core concepts of the ADOxx metamodeling platform, which we use in sections 5 and 6 of this paper for the implementation and evaluation of our approach.

In FDMM, a metamodel **MM** of a modeling language is described using four components, i.e. $\mathbf{MM} = \langle \mathbf{MT}, \preceq, domain, range, card \rangle$. Each metamodel consists of a set model types **MT** which are used to create a set of models **mt**. Each model type $\mathbf{MT_i}$ consists of a set of object types $\mathbf{O_i^T}$, a set of attributes $\mathbf{A_i}$ and a set of datatypes $\mathbf{D_i^T}$. A model type is thus described in FDMM by $\mathbf{MT_i} = \langle \mathbf{O_i^T}, \mathbf{D_i^T A_i} \rangle$. The operator $\preceq$ is an ordering on the set of object types $\mathbf{O_i^T}$ that is used to define an object type hierarchy similar to inheritance hierarchies in object oriented software languages.

*domain* is a function which assigns attributes to object types. The *range* function assigns datatypes, other object types and model types to attributes, while the *card* function defines the cardinality, i.e. the number of allowed values for an attribute. For describing the instantiation of model types to models, object types to objects, and data types to concrete data values, the functions $\mu_{\mathbf{mt}}$, $\mu_{\mathbf{O}}$, and $\mu_{\mathbf{D}}$ are used, which basically take a type from the metamodel as input and map it to a set of instances of this type.

A model $\mathbf{mt_i}$ is an instance of a particular model type and consists of so-called triple statements $\tau_{\mathbf{j}}$ :

[1] http://www.omg.org/spec/OCL/2.0/

$(\mathbf{o_j}\ \mathbf{a_j}\ \mathbf{d_j})$ representing the model content. The first element $\mathbf{o_j}$ of a triple $\tau_\mathbf{j}$ represents an instance of an object type, the second component $\mathbf{a_j}$ represents an attribute and the last component $\mathbf{d_j}$ represents the value for the attribute $\mathbf{a_j}$. The assignment of triples to models is accomplished via the map $\beta : \mathbf{mt} \rightarrow \mathcal{P}(\tau)$. For further details on FDMM especially regarding instantiation functions and additional constraints and correctness criteria we refer to [17].

### 2.2. Concepts of the Semantic Web Rule Language

The specification of the Semantic Web Rule Language (SWRL) is available as a proposal by Horrocks et al. on the the W3C website [21]. SWRL is defined as an ontology - called *SWRL ontology* - which integrates RuleML[2] and OWL. Due to the strong reference to OWL, SWRL can be rather seen as an OWL-extension [22]. SWRL rules have several benefits over existing OWL axioms such as higher expressiveness and less restrictions [7]. The SWRL proposal introduces an XML- and RDF-based syntax [21], whereby it refers to the OWL specification [23] for the corresponding syntax of the underlying OWL elements.

A SWRL rule consists of an antecedent - denoted as *body* - and a consequent - denoted as *head*. Both, the antecedent and the consequent consist of a possibly empty set of atoms, which are connected using conjunctions. Therefore, the order of the atoms belonging either to the consequent or the antecedent has no semantic implication for a rule. The SWRL proposal distinguishes between seven different atoms. All atoms have parameters, which are either OWL constructs – such as references to OWL classes, variables or data values. For example, the *Class* atom of the SWRL specification has a parameter which refers to an OWL class. Also, the *IndidvidualProperty* atom has a parameter which refers to an OWL property.

The proposal for OWL 2 [24] was published a few years later after the release of the SWRL proposal in 2004. Thereby, SWRL rules were not added to the OWL 2 proposal. Rather, the introduction of OWL 2 raised compatibility issues with the SWRL proposal. Glimm et. al. [7] state that the existing rule syntax of the SWRL proposal is "*not very well aligned with the new OWL 2 syntax*". So, they developed an updated SWRL version to align it with the OWL 2 proposal [7]. The updated version is intended as a substitute of the SWRL proposal issued by the W3C [21]. For ensuring decidability, primarily two restrictions are introduced - for more details we refer to [7]: (i) *Safety*: Variables which occur in the consequent have to occur in the antecedent (ii) *DL Safety*: Individuals which are used in rules have to bind to existing individuals of the ontology. Further, each data variable in a data range atom has to occur in a data property atom in the antecedent.

For describing the syntax of the updated SWRL, Glimm et. al. reverted to the generic functional syntax. The updated SWRL - which we will use as foundation for our subsequent elaborations - is widely supported by reasoners, e.g. Pellet [7] and HermiT [25].

Today, a variety of syntaxes for OWL 2 ontologies exist. For example, the current version of Stanford Protégé [26] offers eight different syntaxes for serializing ontologies. However, the OWL-XML syntax of the W3C SWRL proposal [21] is currently not supported by this platform[3]. A good overview of the different OWL syntaxes and their development is given by Horridge in his Ontogenesis blog [27], who stresses the fact that the definition of OWL does not rest on a particular concrete syntax but is rather based on a high level structural specification that is subsequently mapped into different concrete syntaxes. Such a structural description for the OWL 2 is given in [24], where the W3C proposal reverts to the functional syntax - a simple text-based syntax that operates as a bridge to concrete syntaxes [27]. A derivation from the functional to a concrete syntax is then for example described in [28], where the development of the OWL-XML syntax based on OWL 2's functional syntax is explained. Each of the syntaxes has strengths and weaknesses [27]. E.g., the OWL-XML syntax - which we used in our use case - benefits from a strong support of existing XML-tools. Similar to the approaches used for OWL 2, concrete syntaxes can be derived from the functional SWRL syntax specified in [7].

SWRL rules are heavily used in industry and in the scientific community. Recently, SWRL rules were used for language processing [29], food selection systems [30], health care [31], annotation of media data [32, 33], communication protocols [34] and risk management [35]. Applications and frameworks which use or support SWRL rules are e.g. Menthor Editor [36], RuQAR [37], or i-RM [38]. SWRL also served as a starting point for novel rule-based for-

[2] http://ruleml.org/index.html

[3] The files provided at https://www.w3.org/Submission/SWRL/ can therefore not be imported to the Protégé default setup (Status: 17/08/2017)

malisms and approaches. For example, a fuzzy extension for SWRL was introduced in [39]. Jajaga et al. developed a SWRL-based language called C-SWRL for reasoning over stream data [40] and in [41] an approach was introduced for identifying dependencies between SWRL rules.

To the best of the authors knowledge, a comprehensive visual language for SWRL rules has so far not been made available. Although a generic metamodel for supporting the creation of visual rules was introduced in [42], this approach does not support all SWRL atoms. A visual language for SWRL was introduced in [43, 44] which neither supports the import of existing SWRL models nor the different built-in types defined in SWRL. A further implementation presented in [45] neglects export functionalities, the visualization of SWRL atoms as well as OWL constructs. The SWRLTab bundled with Stanford Protégé allows a text-based modification of rules. In the same way, the ORE editor offers a text-based editing of rules[4]. Axiome is a Protégé plugin for the visualization of SWRL rules [46]. Modifications of rules are not supported. Skillen et al. [8] presented a tool for the visual programming of SWRL rules. With a strong focus on usability, that approach does not support all atoms, e.g. it misses the *BuiltIn* atoms. Further, the import of SWRL rules was not considered in this work and the modification of OWL ontologies is not supported. Rule editing using REST webservices was introduced in [6]. In [47] a plugin is presented which allows to create OWL ontologies entering textual rules. As summary of further tools is given in [8].

In summary, our related work analysis shows that a tool for the visual creation and editing of SWRL rules does not exist yet. It seems however favorable to provide such a tool to enhance the usability of SWRL. Although certain attempts have been made in this direction, a more complete and systematic approach for a visual, model-based representation of SWRL is needed.

## 3. A Visual Modeling Approach for SWRL

The visual modeling approach for SWRL that we present in the following is structured along three layers as depicted in Figure 1. The *configuration layer* contains the SWRL model - a visual representation of SWRL rules - as well as the ontologies which are referenced by the SWRL model. The *transformation layer* is responsible for generating the SWRL rules from the SWRL model and for integrating them with the ontology. The result of this step are SWRL rules in a certain syntax, e.g. OWL-XML, including an ontology which can be processed in the *execution layer*.

The ontology can either be stored as a file, as an in-memory data structure or as a visual model. In this section we focus on the visual modeling language for SWRL rules which consists of the (i) syntax, (ii) a visual notation, (iii) and semantics [13].

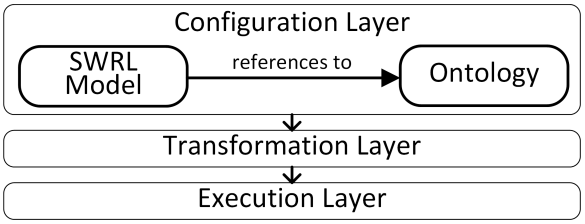



Fig. 1. Layers of the Visual Modeling Approach for SWRL

### 3.1. Syntax

The syntax describes the model elements and valid combinations of them [48]. For the description of the syntax of visual modeling languages, it is often reverted to semi-formal visual metamodels [49, 50].

For the development of the modeling language we tried to balance the trade-offs between expressiveness and usability. For example, a large number of object types simplifies the transformation of the model on the transformation layer. On the contrary, large numbers of object types aggravate modeling. Moody denotes this trade-off as *Graph Economy*, which is further affected by the usage of attributes [51]. Modeling languages without attributes have to provide additional object types as well as references in order to encode the corresponding information. In practice, both approaches exist. Prominent languages such as ArchiMate [52] use only object types without attributes, while other languages such as UML [53] or BPMN [54] employ object types together with attributes extensively. Usually, these attributes are not displayed in models so that modeling tools offer property windows for accessing them - see for example the modeling tools in the OMiLAB.org repository[5].

[4] https://sourceforge.net/projects/ore/files/?source=navbar

[5] See http://www.omilab.org/psm/modelling-tools

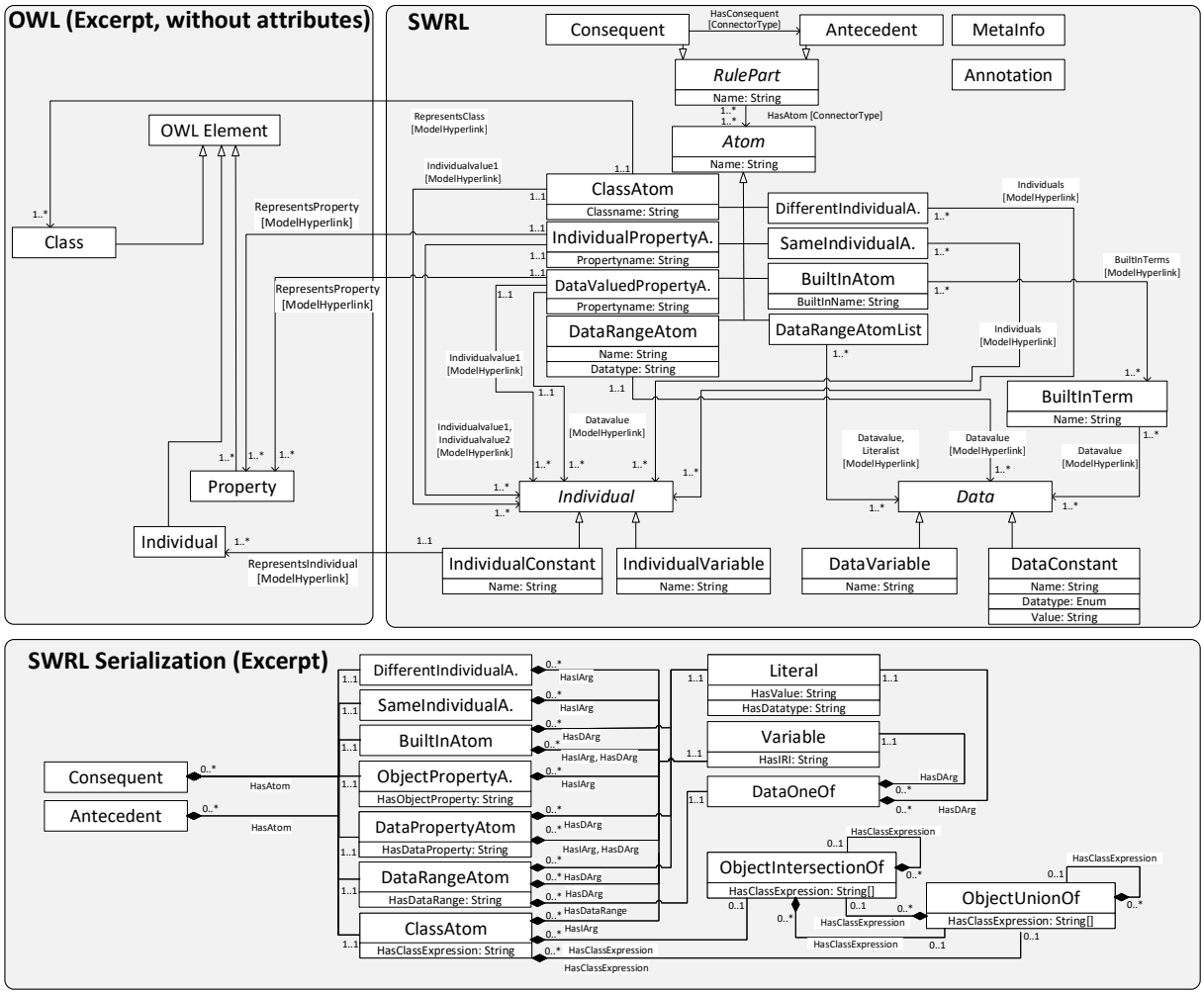



Fig. 2. Excerpt of the SWRL Metamodel Which Is Integrated with the OWL Metamodel, Bottom Shows an Excerpt of the Metamodel of the Serialization

An excerpt of the metamodel of the SWRL modeling language in UML class diagram notation is depicted in Figure 2. It is grouped into three parts: an OWL part, a SWRL part, and a SWRL serialization part. The OWL part and the SWRL part correspond to the configuration layer. The SWRL serialization part in the bottom of the figure belongs to the transformation layer and will be discussed later on. The boxes are UML classes, which are used to represent object types. The classes with names in italics are abstract. These classes do not have a visual representation and can therefore not be instantiated by modelers. The solid connector which has an unfilled triangle ($\triangleleft$) at its end is used to establish a hierarchy between classes similar to inheritance in object oriented programming languages such as Java. It corresponds to the $\preceq$ operator in FDMM as described in section 2.1.

The classes *ClassAtom* or *IndividualPropertyAtom* are therefore sub-classes of the class *Atom*. The UML associations which end with an arrow ($\leftarrow$) and which are annotated with the tag [*ConnectorType*] represent object types that connect other object types. Instances of this type thus stand for edges between objects in a model instance. Finally, the UML associations which are annotated with the tag [*ModelHyperlink*] represent reference attributes. Such reference attributes can be considered as model hyperlinks to other model elements or models. They are not necessarily displayed in a model but are used to navigate between model elements. In the following, we summarize the most

important design decisions for developing the metamodel:

*Classes / Object Types.* To ensure semiotic clarity (c.f. [51]) we mapped each core concept of SWRL to one object type. Therefore, each atom defined in the SWRL standard is represented by a single object type in the visual language and vice versa. Indeed, we splitted the SWRL atom *DataRangeAtom* up in two atoms: *DataRangeAtom* and *DataRangeAtomList*. This is because these two datarange atoms, which contain a datatype or data values, are semantically related, but their syntax is different. Hence, we distinguish between a *DataRangeAtom* object type containing a datatype and a *DataRangeAtomList* object type containing a list of datavalues.

*Attributes.* We decided to use the attributes in our metamodel to reduce the number of object types and thereby enhance usability. In the excerpt of the metamodel depicted in Figure 2, we do not show all the attributes of the object types which we used in order to improve readability. Therefore, only a small set of the actually available attributes are included.

*Terms.* Similar to the generic rule definition metamodel by Brockmans et al. [42], we represented the possible terms - *IndividualVariable*, *IndividualConstant*, *DataVariable* and *DataConstant* - as separate classes / object types, which may be more convenient for model users than a solution based on attributes. Using this approach, syntactical modeling errors can already be avoided at design time. For example, the use of data variables for atoms that do not have data parameters is thereby prohibited.

*Placeholders.* If the ontology underlying a SWRL rule is also to be stored as a visual model, its metamodel has to be integrated with the one for the SWRL. To support this, we integrated the SWRL metamodel with the SeMFIS metamodel [10] for OWL, which we will require for the use case presented later in section 6. We specify model hyperlinks / references from the SWRL metamodel to the OWL metamodel for keeping the SWRL model self contained. This means that the transformation layer should be able to produce valid SWRL rules of the SWRL model, even if no OWL model exists. In total, four classes contain model hyperlinks to the OWL metamodel: (i) *ClassAtom* (ii) *IndividualPropertyAtom* (iii) *DatavaluedPropertyAtom* (iv) *IndividualConstant*. *IndividualPropertyAtom* and *DatavaluedPropertyAtom* atoms may have a model hyperlink to the *Property* object type in the OWL model. The object type *Atom* references an OWL *Class*, which can either be anonymous or non-anonymous. All these elements have an attribute of data type *String* called *Name*. This string is used if no model hyperlink exists. An *IndividualConstant* can represent an OWL *Individual* via the model hyperlink *RepresentsIndividual*. The individual constant can be considered as a placeholder. If the OWL model does not exist, the constant can still be used by the atoms. An alternative to using model hyperlinks to individual constants would be to remove the individual constant from our metamodel and add model hyperlinks from the atoms directly to the OWL model. However this approach has the following drawback: if the OWL model does not exist, the atoms have an empty model hyperlink. The information that two atoms may use the same individual in the OWL model is lost. By using an individual constant object type, the atoms can reference the same individual constant, which in turn can reference an individual in the OWL model.

### *3.2. Notation*

The visual notation of the most important object types is shown in Table 1. For the design of the visual notation we took into account the principles of Moody, in particular *Dual Theory* and *Semantic Transparency* [51]. Dual theory requires that graphical elements are enriched with a textual description. For high semantic transparency we used meaningful symbols or abbreviations for the notation. This resulted in the visual notation shown in Table 1.

For improving the readability of the notation we use dynamic notation. Such a notation changes based on certain attribute states of model elements [12, 55]. I.e., textual information is dynamically added to symbols at run-time, as shown in Figure 4. In the example, the names of the OWL elements are dynamically displayed below the atoms based on the current state of the reference attributes.

### *3.3. Semantics*

The modeling language was developed in order to create SWRL rules. Hence, the semantics of the model elements is identical to the corresponding concepts of the SWRL proposal and the updated version – see [21] and [7] for a detailed semantic description.

In the modeling method itself, no operational SWRL or OWL semantics are included, e.g. in the form of algorithms [56]. It is rather reverted to transformations from the model information to machine-processable representations as will be shown in the next section.

| | | | | | |
|---|---|---|---|---|---|
| Antecedent | The antecedent (condition) of a rule | A C | Class atom, e.g. Person(?x) | A ≠ | Different Individuals Atom, e.g. differentFrom(?x,?y) |
| Consequent | The consequent of a rule | A I | Individual-valued Property Atom, e.g. hasRisk(?x,?y) | A = | Same Individuals Atom, e.g. sameAs(?x,?y) |
| IndividualConstant | Constant representing a data value | A d | Datavalued Property Atom, e.g. hasName(?x,?y) | | MetaInfo for describing meta information about a rule, e.g. about imports |
| IndividualVariable | Variable for storing individual values | A D | Datarange Atom with datatype | TERM | Built-In Term used as reference for composing built-in parameters |
| DataVariable | Variable for storing data values | A DL | Datarange Atom with literal list | | Relation for connecting atoms to antecedents and consequents |
| DataConstant | Constant representing a data value | A B | Built-In Atom, e.g. greaterThan(?age, 17) | | Relation for connecting antecedents and consequents |

Table 1
Excerpt of the Elements Used for the SWRL Modeling Language [9]

## 4. Transformation of SWRL Models

The transformation layer is responsible for generating machine-processable SWRL rules from the SWRL models on the configuration layer. Therefore, transformation rules as well as a transformation engine is necessary as depicted in Figure 3. The transformation rules describe how to transform the models to a concrete syntax. The transformation engine processes the models according to the given transformation rules.

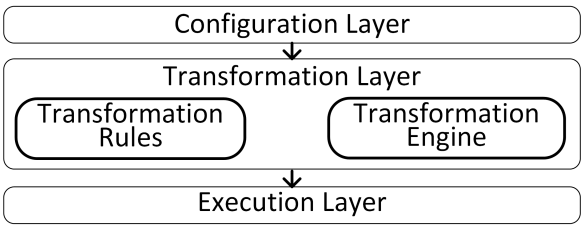



Fig. 3. Transformation Rules and the Transformation Engine as Parts of the Transformation Layer

An example transformation is shown in Figure 4, which represents Example 5.1-1 of the SWRL proposal [21]. The arrows between the atoms (round symbols with the letter 'A') and the properties (yellow symbols with the letter 'P') represent the model hyperlinks, which were added to the figure in order to improve readability. The visual SWRL rule is identical to the SWRL rule in the textual notation shown at the bottom of the figure. Such a textual notation is typically used on platforms such as Stanford Protégé. Each atom of the visual SWRL rule is transformed to the corresponding string in the textual SWRL rule.

In the following, we describe the core transformation rules for transforming visual SWRL models. Therefore, we first define both, the SWRL model as well as the serialization using a formalism. Based on the description, the transformation rules are introduced. The description of the serialization is based on the functional rule syntax of [7] and therefore independent of concrete syntaxes. With a focus on SWRL, we do not describe the transformation of OWL constructs such as OWL individuals or OWL properties in the following subsections. We also neglect the description of exception cases where e.g. referenced OWL concepts are missing in the model. However, we support these transformations in our prototype described in section 6.

A description of the SWRL modeling language as well as it's serialization using FDMM is given in Appendix A and B. For the following transformation rules we will only require a subset of this formalization. In

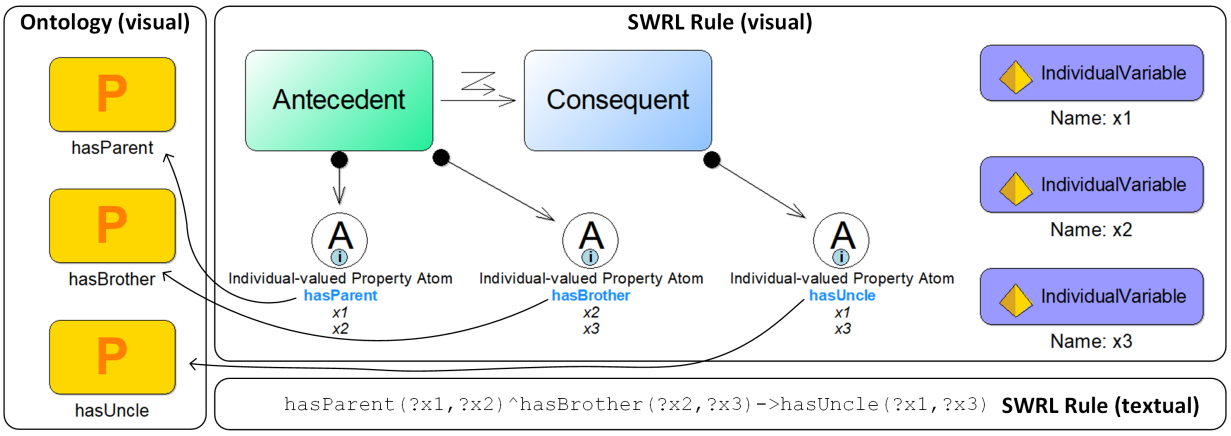



Fig. 4. Transformation of a Visual SWRL Rule to a Textual SWRL Rule

particular, we will refer to the model type for OWL as $\mathbf{MT}_{OWL}$ and the model type for the visual SWRL rules as $\mathbf{MT}_{SWRL}$. In addition, the serialization is also represented in FDMM via the model type $\mathbf{MT}_{\overline{SWRL}}$.

For navigating to an attribute, the FDMM formalism requires the use of triple statements. For example, consider an instance of an OWL *Class* $o_{Class} \in \mu_{\mathbf{o}}(Class, \mathbf{MT}_{OWL})$ and an instance of *ClassAtom* $o_{ClassAtom} \in \mu_{\mathbf{o}}(ClassAtom, \mathbf{MT}_{SWRL})$. The *ClassAtom* instance shall now be part of an instance of a SWRL model $m_{SWRL} \in \mathbf{MT}_{SWRL}$ and reference the OWL *Class* instance, i.e. $(o_{ClassAtom}\ RepresentsClass\ o_{Class}) \in \beta(m_{SWRL})$. It shall also be ensured that for the OWL *Class* instance, the attribute *IsAnonymous* is set to the value 'Y' - which indicates to the user that the class instance is anonymous. I.e., $(o_{Class}\ IsAnonymous\ 'Y') \in \beta(m_{OWL})$.

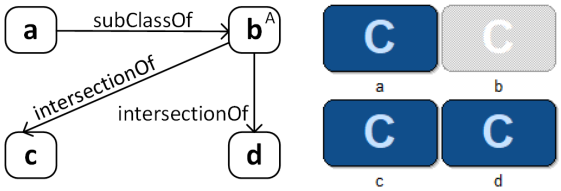
In the following we use an abbreviation for the navigation via triples by using a dot syntax similar to the Object Constraint Language (OCL) in UML. Thereby, the previously described two triples are summarized as follows: $o_{ClassAtom}$.RepresentsClass.IsAnonmyous='Y': the first element is an instance of an object type, while the elements are attribute names which are used to access the attribute value of the last attribute.

For the description of the rules, we will use the following arrow types. The left side of the arrow $\Rightarrow$ shows the input value which the rule receives while the right side of the arrow shows the return value of the rule. The arrow $\mapsto$ is used as an operator to assign values to attributes. If the attribute is a set, then the value on the left side is added to the set. Even if the rules are independent from its concrete serialization, we use OWL-XML serialization examples to improve understandability.

### 4.1. Transformation Rules

When designing transformation rules for the serialization of the visual SWRL models to OWL, some differences have to be taken into account beforehand. This concerns the following four aspects: i. Anonymous Classes, ii. Model Hyperlinks, iii. BuiltIn Terms, and iv. Variables. This will be discussed in more detail in the following.

(i) **Anonymous Classes.** In OWL ontologies, anonymous classes are classes without a name. In the model type $\mathbf{MT}_{OWL}$, *Class* elements have an attribute *IsAnonymous* which indicates whether the model element representing an OWL class is anonymous. An example is depicted in Figure 5a which shows that class *a* is a subclass of *b*. Class $b^A$ is anonymous and defined as an *intersection of* the classes *c* and *d*.




(a) Schematic Overview of the Model Elements (Class b Is Anonymous)

(b) Model of an OWL Ontology in SeMFIS (Class b Is Anonymous)

Fig. 5. OWL Class Hierarchy Schematic (Left) and in the Model (Right)

Figure 5b shows the corresponding visual model of the OWL ontology using the SeMFIS modeling language for OWL [10]. Thereby, the class b is anonymous. Such properties are typically set by the user by

reverting to property windows in a modeling tool. We will use this for the prototypical implementation of the modeling language – see Figure 9. In the serialization $\mathbf{MT}_{\overline{SWRL}}$, such an attribute however does not exist. Hence, as shown for the corresponding serialization in OWL-XML in Listing 1, the anonymous class does not appear any more. Instead, its content is copied directly to the position where it is referenced. However, such an anonymous class could e.g. be referenced in *ClassAtom* elements.

(ii) **Model Hyperlinks.** While the visual model makes use of model hyperlinks, e.g. for connecting OWL classes with class atoms or for defining hierarchies as described in the previous example, such hyperlinks do not exist in the serialization. For connecting e.g. OWL classes with class atoms, the *ClassAtom* element links to the corresponding OWL *Class* element via a reference attribute type.

(iii) **Variables.** While in the model it is distinguished between different variables - *IndividualVariable* and *DataVariable* - the serialization does not distinguish between them[6]. In the visual model, the variables are defined using separate object types to which the atoms refer using model hyperlinks. On the contrary, in the serialization variables are not defined separately - they are used directly in the atoms.

(iv) **BuiltIn Terms.** The visual model provides a separate model element for *BuiltIn* terms to define an order of the terms. Similarly to the variables, the terms are not separately defined in the serialization, but they are used directly in the *BuiltIn* atom.

**LISTING 1:** Example for the Serialization of an Anonymous Class in OWL-XML using the Subclass Axiom

```
<owlx:SubClassOf / >
    <owlx:Class IRI= a / >
    <owlx:ObjectIntersectionOf >
        <owlx:Class IRI= c / >
        <owlx:Class IRI= d / >
    </owlx:ObjectIntersectionOf>
</owlx:SubClassOf>>
```

**Serialization of SWRL Models.** With these considerations we can now advance to the transformation from visual SWRL models to a serialization. Thereby we focus on the description of the transformation of SWRL constructs. We will neglect OWL constructs such as individuals in the following rules.

The following rule (**R1**) describes the serialization of a *ClassAtom* element in a visual SWRL model:

**R1. ClassAtom - Rule:** $m \in \mu_{\mathbf{o}}(ClassAtom, \mathbf{MT}_{SWRL})$ $\Rightarrow$ s $\in \mu_{\mathbf{o}}(ClassAtom, \mathbf{MT}_{\overline{SWRL}})$:
**if** (m.RepresentsClass.IsAnonymous='Y') {
  **if** ( | m.RepresentsClass.IntersectionOf $\cup$
  m.RepresentsClass.UnionOf | =1) {
    **R9**(*m.RepresentsClass*)
    $\mapsto$ s.HasClassExpression,
    **R14**(*m.RepresentsClass*)
    $\mapsto$ s.HasClassExpression
  }
  **else** {
    **R12**(*m.RepresentsClass*) $\mapsto$
    s.HasClassExpression
  }
}
**else** {
  m.RepresentsClass.Name $\mapsto$ s.HasClassExpression
},
**R13**(*m.Individualvalue*1.*Name*) $\mapsto$ s.HasIArg

If the referenced OWL *Class* is anonymous, then the content of the anonymous class becomes part of the class atom in the serialization. On the contrary, if the referenced class is not anonymous, then the corresponding OWL class is referenced in the serialization using its IRI. During the serialization, references to anonymous classes are usually resolved and the content of the anonymous class is inserted at the position it was referenced. However, complex anonymous classes require a special handling. For example, the ontology depicted in Figure 6 results in the serialization shown in Listing 2 - assuming that the anonymous class $b^A$ is referenced by a *ClassAtom* element in the model.

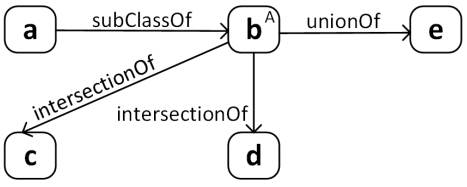



Fig. 6. Schematic Overview of the Model Elements (Class $b^A$ Is Anonymous)

[6]The paper introducing the syntax we use here [7] is available at http://webont.org/owled/2009/papers/owled2009_submission_16.pdf were it does not distinguish between the variable, while the paper available at https://www.cs.ox.ac.uk/files/2445/rulesyntaxTR.pdf does make a distinction.

As class atoms, but also *complement of* and *sub class of* class description axioms are allowed to contain only a single axiom such as *intersection of* or *union of* [7, 21], the resolving strategy which puts the content of an anonyomous class to the place it was referenced leads to invalid serializations - the class atom shown in listing 2 contains two axioms.

**LISTING 2:** Serialization of a Class atom

```
<owlx:ClassAtom / >
    <owlx:ObjectIntersectionOf >
        <owlx:Class IRI= c / >
        <owlx:Class IRI= d / >
    </owlx:ObjectIntersectionOf>
    <owlx:ObjectUnionOf >
        <owlx:Class IRI= e / >
    </owlx:ObjectUnionOf>
    [...]
</owlx:ClassAtom>>
```

This issue originates from a structural problem: anonyomous classes can have multiple axioms but a class atom can only contain a single axiom. In order to create valid SWRL serializations even in such scenarios, we wrap the content of anonymous classes into an *intersection of* axiom in cases in which the class contains several class description axioms. However, this *wrapping* leads to a loss of information.

The following rule describes the transformation of the *IndividualProperty* atom. No anonymous properties are considered in the existing OWL metamodel.

**R2.IndividualPropertyAtom - Rule:** $m \in \mu_{\mathbf{o}}($ *IndividualPropertyAtom*, $\mathbf{MT}_{SWRL}) \Rightarrow$ s $\in \mu_{\mathbf{o}}($ *ObjectPropertyAtom*, $\mathbf{MT}_{\overline{SWRL}})$:
m.RepresentsProperty.Name $\mapsto$ s.HasObjectProperty,
**R13**(*m.Individualvalue*1.*Name*) $\mapsto$ s.HasIArg,
**R13**(*m.Individualvalue*2.*Name*) $\mapsto$ s.HasIArg

For the serialization of the *DatavaluedProperty* atom, the data values need to be serialized either to a variable or to a literal:

**R3.DatavaluedPropertyAtom - Rule:** $m \in \mu_{\mathbf{o}}($ *DatavaluedPropertyAtom*, $\mathbf{MT}_{SWRL}) \Rightarrow$ s $\in \mu_{\mathbf{o}}($ *DatavaluedPropertyAtom*, $\mathbf{MT}_{\overline{SWRL}})$:
m.RepresentsProperty.Name $\mapsto$ s.HasDataProperty,
**R13**(*m.Individualvalue*1.*Name*) $\mapsto$ s.HasIArg,
**R11**(*m.Datavalue*) $\mapsto$ s.HasDArg

The serialization of the *DataRange* atom is similar to the one of the *DatavaluedProperty* atom. However, the data type has to be considered:

**R4.DataRangeAtom - Rule:** $m \in \mu_{\mathbf{o}}($ *DataRangeAtom*, $\mathbf{MT}_{SWRL}) \Rightarrow$ s $\in \mu_{\mathbf{o}}($ *DataRangeAtom*, $\mathbf{MT}_{\overline{SWRL}})$:
**R11**(m.Datavalue) $\mapsto$ s.HasDArg
m.Datatype $\mapsto$ s.HasDataRange

For the *DifferentIndividuals* atom all individuals of the list *individuals* have to be serialized:

**R5.DifferentIndividualsAtom - Rule:** $m \in \mu_{\mathbf{o}}($ *DifferentIndividualsAtom*, $\mathbf{MT}_{SWRL}) \Rightarrow$ s $\in \mu_{\mathbf{o}}($ *DifferentIndividualsAtom*, $\mathbf{MT}_{\overline{SWRL}})$:
{ $\forall$ i $\in$ m.Individuals:**R13**(i.Name)} $\mapsto$ s.HasIArg

The transformation rule for the *SameIndividuals* atom is identical to the one of the *DifferentIndividuals* atom:

**R6.SameIndividualsAtom - Rule:** $m \in \mu_{\mathbf{o}}($ *SameIndividualsAtom*, $\mathbf{MT}_{SWRL}) \Rightarrow$ s $\in \mu_{\mathbf{o}}($ *SameIndividualsAtom*, $\mathbf{MT}_{\overline{SWRL}})$:
{ $\forall$ i $\in$ m.Individuals:**R13**(i.Name)} $\mapsto$ s.HasIArg

The model element *DataRangeAtomList* was introduced to simplify modeling - it is a special type of the *DataRangeAtom*:

**R7.DataRangeAtomList - Rule:** $m \in \mu_{\mathbf{o}}($ *DataRangeAtomList*, $\mathbf{MT}_{SWRL}) \Rightarrow$ s $\in \mu_{\mathbf{o}}($ *DataRangeAtom*, $\mathbf{MT}_{\overline{SWRL}})$:
**R11**(m.Datavalue) $\mapsto$ s.HasDArg
**R15**(m) $\mapsto$ s.HasDataRange

The *BuiltIn* atom makes use of special term elements in the model:

**R8.BuiltInAtom - Rule:** $m \in \mu_{\mathbf{o}}($ *BuiltInAtom*, $\mathbf{MT}_{SWRL}) \Rightarrow$ s $\in \mu_{\mathbf{o}}($ *BuiltInAtom*, $\mathbf{MT}_{\overline{SWRL}})$:
m.BuiltInName $\mapsto$ s.HasIRI,
{ $\forall$ i $\in$ m.BuiltInTerms:**R11**(i.Datavalue)}
$\mapsto$ s.HasDArg

The following rule is called within other rules for the serialization of OWL *Class* elements:

**R9.OWLClass - Rule:** $m \in \mu_{\mathbf{o}}($ *OWLClass*, $\mathbf{MT}_{OWL})$
$\Rightarrow$ s $\in \mu_{\mathbf{o}}($ *ObjectIntersectionOf*, $\mathbf{MT}_{\overline{SWRL}})$:

$\{\forall i \in$ m.IntersectionOf | i.IsAnonymous='N' : **R10**$(i)\}$
$\mapsto$ s.HasClassExpression,
$\{\forall i \in$ m.IntersectionOf | i.IsAnonymous='Y': **R9**$(i)\}$
$\mapsto$ s.HasClassExpression,
$\{\forall i \in$ m.IntersectionOf | i.IsAnonymous='Y' : **R14**$(i)\}$
$\mapsto$ s.HasClassExpression

This rule is called to resolve the referenced OWL classes, e.g. in intersection of axioms:

**R10. OWLClass String Reference Helper - Rule:**
$m \in \mu_{\mathbf{o}}(OWLClass, \mathbf{MT}_{OWL}) \Rightarrow$ s $\in String$:
m.Name $\mapsto$ s

With the following rule a helper rule for creating variables and literals is provided:

**R11. DataItem Helper - Rule:** $m \in (\mu_{\mathbf{o}}(DataConstant, \mathbf{MT}_{SWRL}) \cup \mu_{\mathbf{o}}(DataVariable, \mathbf{MT}_{SWRL}))$
$\Rightarrow$ s $\in (\mu_{\mathbf{o}}(Literal, \mathbf{MT}_{\overline{SWRL}}) \cup \mu_{\mathbf{o}}(Variable, \mathbf{MT}_{\overline{SWRL}}))$:
**if** (m.Value=$\varnothing$) {
  m.Name $\mapsto$ s.HasIRI
  s $\in \mu_{\mathbf{o}}(Variable, \mathbf{MT}_{\overline{SWRL}})$
}
**else** {
  m.Value $\mapsto$ s.HasValue
  m.Datatype $\mapsto$ s.HasDatatype
  s $\in \mu_{\mathbf{o}}(Literal, \mathbf{MT}_{\overline{SWRL}})$
}

This rule is called for creating a wrapping *intersection of* axiom for other axioms:

**R12.OWLClass - Rule:** $m \in \mu_{\mathbf{o}}(\ OWLClass, \mathbf{MT}_{OWL})$
$\Rightarrow$ s $\in \mu_{\mathbf{o}}(\ ObjectIntersectionOf, \mathbf{MT}_{\overline{SWRL}})$:
**R9**$(m) \mapsto$ s.HasClassExpression
**R14**$(m) \mapsto$ s.HasClassExpression

This Rule is used for creating a variable used within atoms:

**R13.Variable - Rule:** $m \in$ String
$\Rightarrow$ s $\in \mu_{\mathbf{o}}(\ Variable, \mathbf{MT}_{\overline{SWRL}})$:
m $\mapsto$ s.HasIRI

The following rule is called within other rules for the serialization of the OWL *Class* elements:

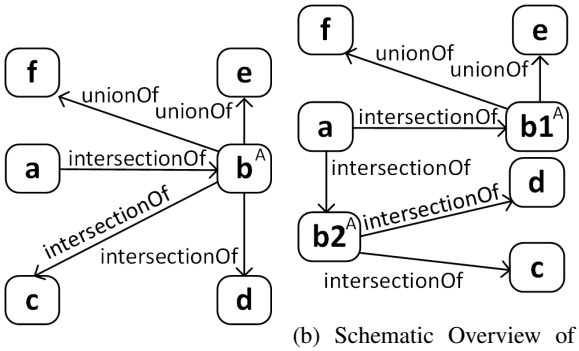
**R14.OWLClass - Rule:** $m \in \mu_{\mathbf{o}}(\ OWLClass, \mathbf{MT}_{OWL})$
$\Rightarrow$ s $\in \mu_{\mathbf{o}}(\ ObjectUnionOf, \mathbf{MT}_{\overline{SWRL}})$:
$\{\forall i \in$ m.UnionOf | i.IsAnonymous='N' : **R10**$(i)\}$
$\mapsto$ s.HasClassExpression,
$\{\forall i \in$ m.UnionOf | i.IsAnonymous='Y': **R9**$(i)\}$
$\mapsto$ s.HasClassExpression,
$\{\forall i \in$ m.UnionOf | i.IsAnonymous='Y': **R14**$(i)\}$
$\mapsto$ s.HasClassExpression

The following rule is called for creating a list of values used in *DataRangeAtoms*:

**R15.DataOneOf - Rule:** $m \in \mu_{\mathbf{o}}(\ DataRangeAtomList, \mathbf{MT}_{SWRL}) \Rightarrow$ s $\in \mu_{\mathbf{o}}(\ DataOneOf, \mathbf{MT}_{\overline{SWRL}})$:
$\{\ \forall$ i $\in$ m.Literallist:**R11**(i) $\} \mapsto$ s.HasDArg

**Deserialization to SWRL Models.** When rule descriptions that already exist in a serialized format shall be transformed into visual models, further transformation rules for this *deserialization* are required. These will be described in the following.

Due to the serialization, a loss of information occurs: the references to anonymous classes are replaced by the content of the class. If an anonymous class is called several times, its content is inserted several times in the serialization. In the serialization it is not possible to find out if two identical anonymous code sections result from the same class or from two different anonymous classes of the visual model. Further, it is not possible to find out if two class description axioms such as *intersection of* and *union of*, which are siblings (i.e. they belong to the same hierarchical level), occur from a single anonymous class or from two anonymous classes in the visual model.




(a) Schematic Overview of the Model Elements (Class $b^A$ Is Anonymous)

(b) Schematic Overview of the Model Elements (Class $b1^A$ and $b2^A$ Are Anonymous)

Fig. 7. Schematic OWL Class Relations (of the Visual Model) with Identical Serializations

An example is depicted in Figure 7. The classes $b^A$, $b1^A$ and $b2^A$ are anonymous. The serialization of both on-

tologies is identical - see the correponding Listing 4, which will be discussed later on. Given the serialization, it is not possible to identify the original model from which the serialization was generated as both models are valid for generating the given serialization. Therefore, we distinguish between two strategies for deserializing classes. These are denoted *Non-anonymous Class Deserialization Strategy* and *Anonymous Class Deserialization Strategy*.

*(i)* ***Non-anonymous Class Deserialization Strategy.*** This strategy is used for deserializing non-anonymous classes to class elements of the visual model. We use an example for describing this strategy. Consider the serialization given in Listing 3. Class $a$ is a non-anonymous class as it has a name (attribute IRI). The deserialization strategy creates for this serialization a visual model with a class element $a$ representing the non-anonymous OWL class. It has model hyperlinks to the other classes to which it is related with the equivalent class axioms. For the given example, class $a$ has hyperlinks to the classes $b$ and $c$ which represent the *intersection of* axiom. This strategy is also applied for other axioms such as union axioms or subclass axioms.

However, if the serialization has two identical axioms[7] – such as the union axiom in the given example – then two anonymous classes are created in the model in order to avoid blurring the intended meaning. Instead of having direct hyperlinks to the classes $d$, $e$, $f$, and $g$, the class element $a$ has hyperlinks to two anonymous class elements and the anonymous class elements reference to the classes $d$, $e$, $f$, and $g$. Hence, the resulting model element which represents class $a$, has a hyperlink to the two anonymous classes $i$, $j$ which point to the model elements contained in the axioms: $a, i, j, d, e, f, g \in \mu_{\mathbf{o}}(OWLClass, \mathbf{MT}_{OWL})$, $a$.UnionOf = $\{i, j\}$, $i$.UnionOf = $\{d, e\}$, $j$.UnionOf = $\{f, g\}$.

*(ii)* ***Anonymous Class Deserialization Strategy.*** We identified two different resolution strategies for anonymous classes which we want to illustrate using the serialization shown in Listing 4. The content of the first *ObjectIntersectionOf* element contains anonymous classes - class description axioms within other class description axioms.

We identified two fundamental strategies for their deserialization: a) ***Single Class Strategy***. This strategy creates a single anonymous class $i$ for the class description axioms which occur in other class description axioms. So we get a class element which has two lists of model hyperlinks representing the intersection axiom as well as the union axiom: i,a,c,d,e $\in \mu_{\mathbf{o}}(\ OWLClass, \mathbf{MT}_{OWL})$, $a$.IntersectionOf = $\{i\}$, $i$.UnionOf = $\{e, f\}$, $i$.IntersectionOf = $\{c, d\}$. In other words, the strategy groups the axioms to a single class. Thereby, it makes the implicit assumption that all axioms belong to a single class. b) ***Multiple Class Strategy***. Unlike the single class strategy, this strategy does not group the axioms to a single class. For each axiom, a separate anonymous class is created. So, for the given listing, two anonymous classes $i$, $j$ are created: $i, j, a, c, d, e, f \in \mu_{\mathbf{o}}(OWLClass, \mathbf{MT}_{OWL})$, $a$.IntersectionOf = $\{i, j\}$, $i$.UnionOf = $\{e, f\}$, $j$.IntersectionOf = $\{c, d\}$.

As the example illustrates, the multiple class strategy usually results in the creation of more anonymous classes than the single class strategy. In the following rules, we revert to the multiple class strategy as it does not make any assumptions regarding the grouping.

**LISTING 3:** Non-Anonymous Class Deserialization Strategy

```
<owlx:Declaration >
    <owlx:Class IRI= a / >
</owlx:Declaration>
[...]
<owlx:EquivalentClasses >
    <owlx:Class IRI= a / >
    <owlx:ObjectIntersectionOf >
        <owlx:Class IRI= b / >
        <owlx:Class IRI= c / >
    </owlx:ObjectIntersectionOf>
</owlx:EquivalentClasses>
<owlx:EquivalentClasses >
    <owlx:Class IRI= a / >
    <owlx:ObjectUnionOf >
        <owlx:Class IRI= d / >
        <owlx:Class IRI= e / >
    </owlx:ObjectUnionOf>
</owlx:EquivalentClasses>
<owlx:EquivalentClasses >
    <owlx:Class IRI= a / >
    <owlx:ObjectUnionOf >
        <owlx:Class IRI= f / >
        <owlx:Class IRI= g / >
    </owlx:ObjectUnionOf>
</owlx:EquivalentClasses>
```

[7] Two axioms with the same name and which belong to the same class.

**LISTING 4:** Anonymous Class Deserialization Strategy

```
<owlx:EquivalentClasses >
    <owlx:Class IRI= a / >
    <owlx:ObjectIntersectionOf >
        <owlx:ObjectIntersectionOf >
            <owlx:Class IRI= c / >
            <owlx:Class IRI= d / >
        </owlx:ObjectIntersectionOf>
        <owlx:ObjectUnionOf >
            <owlx:Class IRI= e / >
            <owlx:Class IRI= f / >
        </owlx:ObjectUnionOf>
    </owlx:ObjectIntersectionOf>
</owlx:EquivalentClasses>
```

Our metamodel distinguishes between four different types of terms. However, in the serialization only variables can be explicitly defined without distinguishing between individual variables and data variables. The atom which contains the variable has to be analyzed in order to find out if the variable is an individual variable or a data variable. If e.g. the defined variable is used as individual in a class atom, then this variable is an individual variable. If the variable is used in a *BuiltIn* atom, then the variable is a data variable.

The following rule describes the transformation of the *Class* atom. The serialized class atom either refers to an existing OWL class or it contains an anonymous class. The function rand() stands for a function that generates a random, unique string similar to the concept of UUIDs, which is used to distinguish the created model elements.

**R1. ClassAtom - Rule:** s $\in \mu_{\mathbf{o}}(ClassAtom, \mathbf{MT}_{\overline{SWRL}})$
$\Rightarrow m \in \mu_{\mathbf{o}}(ClassAtom, \mathbf{MT}_{SWRL})$:
rand() $\mapsto$ m.Name,
**R11**$(s.HasClassExpression) \mapsto$ m.RepresentsClass,
**R12**$(s.HasIArg) \mapsto$ m.Individualvalue1

The following rule describes the deserialization of an *IndividualProperty* atom. Thereby, the individuals are transformed to variables due to the negligence of individual constants. Further, the reference to the property - using the name of the property - is resolved.

**R2.IndividualPropertyAtom - Rule:** s $\in \mu_{\mathbf{o}}($
$IndividualPropertyAtom, \mathbf{MT}_{\overline{SWRL}}) \Rightarrow m \in \mu_{\mathbf{o}}($
$IndividualPropertyAtom, \mathbf{MT}_{SWRL})$ :
rand() $\mapsto$ m.Name,
**R13**(s.HasObjectProperty) $\mapsto$ m.RepresentsProperty,
**R12**(s.HasIArg.at(0)) $\mapsto$ m.Individualvalue1,
**R12**(s.HasIArg.at(1)) $\mapsto$ m.Individualvalue2

For the serialization of the *DatavaluedProperty* atom, the datavalues need to be serialized either to a variable or to a literal:

**R3.DatavaluedPropertyAtom - Rule:** s $\in \mu_{\mathbf{o}}($
$DataPropertyAtom, \mathbf{MT}_{\overline{SWRL}}) \Rightarrow m \in \mu_{\mathbf{o}}($
$DatavaluedPropertyAtom, \mathbf{MT}_{SWRL})$:
rand() $\mapsto$ m.Name,
**R13**(s.HasDataProperty) $\mapsto$ m.RepresentsProperty
**R12**(s.HasIArg) $\mapsto$ m.Individualvalue1
**if** (s.HasDArg $\in \mu_{\mathbf{o}}($ $Literal, \mathbf{MT}_{\overline{SWRL}})$) {
  **R15**(s.HasDArg) $\mapsto$ m.Datavalue
}
**else** {
  **R14**(s.HasDArg) $\mapsto$ m.Datavalue
}

The introduced modeling language distinguishes between *DataRangeAtom* and *DataRangeAtomList*. Therefore, the following rule checks if a datatype exists in order to make a correct transformation:

**R4.DataRangeAtom - Rule:** s $\in \mu_{\mathbf{o}}(DataRangeAtom,$
$\mathbf{MT}_{\overline{SWRL}}) \Rightarrow m \in \mu_{\mathbf{o}}(DataRangeAtom, \mathbf{MT}_{SWRL})$
$\vee\ \mu_{\mathbf{o}}(DataRangeAtomList, \mathbf{MT}_{SWRL})$:
rand() $\mapsto$ m.Name,
**if** (s.HasDArg $\in \mu_{\mathbf{o}}(Literal, \mathbf{MT}_{\overline{SWRL}})$)) {
  **R15**$(s.HasDArg) \mapsto$ m.Datavalue
}
**else** {
  **R14**$(s.HasDArg) \mapsto$ m.Datavalue
}
**if** (s.HasDataRange $\in \mu_{\mathbf{o}}(DataOneOf, \mathbf{MT}_{\overline{SWRL}})$ ) {
  { $\forall$ i $\in$ s.HasDataRange.HasDArg:
    **if** (i $\in \mu_{\mathbf{o}}(Literal, \mathbf{MT}_{\overline{SWRL}})$) {
      **R14**$(i)$
    }
    **R15**$(i)$
  } $\mapsto$ m.Literallist,
  m $\in \mu_{\mathbf{o}}(DataRangeAtomList, \mathbf{MT}_{SWRL})$
}
**else** {
  s.HasDataRange $\mapsto$ m.Datatype
  m $\in \mu_{\mathbf{o}}(DataRangeAtom, \mathbf{MT}_{SWRL})$
}

For the *DifferentIndividuals* atom all individuals have to be serialized:

**R5.DifferentIndividualsAtom - Rule:** s $\in \mu_{\mathbf{o}}($ *DataRangeAtom*, $\mathbf{MT}_{\overline{SWRL}}) \Rightarrow m \in \mu_{\mathbf{o}}($ DataRangeAtom ,$\mathbf{MT}_{SWRL})$:
rand() $\mapsto$ m.Name,
{ $\forall$ i $\in$ s.HasIArg:**R12**$(i)$} $\mapsto$ m.Individuals

The transformation rule for the *SameIndividuals* atom is identical to the one of the *DifferentIndividuals* atom:

**R6.SameIndividualsAtom - Rule:** s $\in \mu_{\mathbf{o}}($ *SameIndividualsAtom*, $\mathbf{MT}_{\overline{SWRL}}) \Rightarrow m \in \mu_{\mathbf{o}}($ *SameIndividualsAtom*, $\mathbf{MT}_{SWRL})$:
s.Name $\mapsto$ m.Name,
{ $\forall$ i $\in$ s.HasIArg:**R12**$(i)$} $\mapsto$ m.Individuals

The *BuiltIn* atom makes use of special term elements in the model which are transformed separately:

**R7.BuiltInAtom - Rule:** s $\in \mu_{\mathbf{o}}($*BuiltInAtom*, $\mathbf{MT}_{\overline{SWRL}}) \Rightarrow m \in \mu_{\mathbf{o}}($ *BuiltInAtom*, $\mathbf{MT}_{SWRL})$:
rand() $\mapsto$ m.Name,
s.HasIRI $\mapsto$ m.BuiltInName,
{ $\forall$ i $\in$ s.HasDArg:**R10**(i)} $\mapsto$ m.BuiltInTerms }

The following rule is a helper rule for creating *BuiltIn* terms:

**R10.BuildInTerms Helper - Rule:** s $\in$ $(\mu_{\mathbf{o}}($ *Literal*, $\mathbf{MT}_{\overline{SWRL}}) \cup \mu_{\mathbf{o}}($*Variable*, $\mathbf{MT}_{\overline{SWRL}}))$
$\Rightarrow m \in \mu_{\mathbf{o}}($*BuiltInTerm* $\mathbf{MT}_{SWRL})$:
**if** (s $\in \mu_{\mathbf{o}}($*Literal*, $\mathbf{MT}_{\overline{SWRL}})$) {
  **R15**(s) $\mapsto$ m.Datavalue
}
**else** {
  **R14**(s) $\mapsto$ m.Datavalue
}

The following rule is a helper rule which creates either a non-anonymous class or an anonymous class. Latter is done in the empty else branch - we neglected it here with a focus on the serialization/deserialization of SWRL constructs:

**R11.Class Helper - Rule:** s $\in$ $\mu_{\mathbf{o}}($*ObjectUnionOf*, $\mathbf{MT}_{\overline{SWRL}}) \cup \mu_{\mathbf{o}}($*ObjectIntersectionOf*, $\mathbf{MT}_{\overline{SWRL}}) \cup$ String $\Rightarrow m \in \mu_{\mathbf{o}}($ *OWLClass*, $\mathbf{MT}_{OWL})$:
**if** (s $\in \mu_{\mathbf{o}}($*ObjectIntersectionOf*, $\mathbf{MT}_{\overline{SWRL}}) \vee$
s $\in \mu_{\mathbf{o}}($*ObjectUnionOf*, $\mathbf{MT}_{\overline{SWRL}})$) {
  'Y' $\mapsto$ m.IsAnonymous,
  rand() $\mapsto$ m.Name,
  **if** (s $\in \mu_{\mathbf{o}}($*ObjectIntersectionOf*, $\mathbf{MT}_{\overline{SWRL}})$) {
    { $\forall$ i $\in$ s.HasClassExpression:**R11**(i)}
    $\mapsto$ m.IntersectionOf
  } **else** {
    { $\forall$ i $\in$ s.HasClassExpression:**R11**(i)}
    $\mapsto$ m.UnionOf
  }
} **else** {
}

The following transformation rule describes the creation of a variable that is created based on a name:

**R12.Individualvariable Helper - Rule:**
s $\in \mu_{\mathbf{o}}($*Variable*, $\mathbf{MT}_{\overline{SWRL}})$,
$\Rightarrow m \in \mu_{\mathbf{o}}($*IndividualVariable*, $\mathbf{MT}_{SWRL})$:
s.HasIRI $\mapsto$ m.Name
//individual constant neglected

This rule creates or finds an OWL *property* based on a given string which contains the string:

**R13.Property Helper - Rule:** s $\in$ *String*,
$\Rightarrow m \in \mu_{\mathbf{o}}($*OWLProperty*, $\mathbf{MT}_{SWRL})$:
  //refer to OWL Property
}

The following transformation rule describes the creation of a variable which is created based on a name:

**R14.Datavariable Helper - Rule:** s $\in \mu_{\mathbf{o}}($ *Variable* , $\mathbf{MT}_{\overline{SWRL}})$
$\Rightarrow m \in \mu_{\mathbf{o}}($*DataVariable*, $\mathbf{MT}_{SWRL})$:
s.HasIRI $\mapsto$ m.Name

The following rule creates a dataconstant whereby the name is created randomly:

**R15.Dataconstant Helper - Rule:** s $\in \mu_{\mathbf{o}}($ *Literal*, $\mathbf{MT}_{\overline{SWRL}}) \Rightarrow m \in \mu_{\mathbf{o}}($*DataConstant*, $\mathbf{MT}_{SWRL})$:
rand() $\mapsto$ m.Name
s.HasValue $\mapsto$ m.Value
s.HasDatatype $\mapsto$ m.Datatype

## 5. Prototypical Technical Implementation

In order to evaluate the feasibility of the presented approach, we created a prototypical technical implementation. This comprised the implementation of the visual modeling language as well as the transformation rules for the serialization and deserialization of the models. Figure 8 shows technologies and syntaxes used for the implementation.

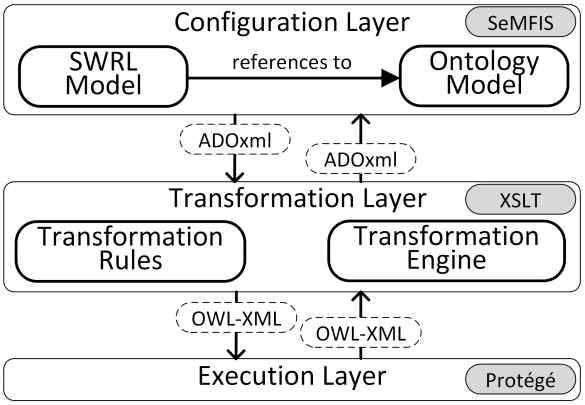



Fig. 8. Technologies Used for Implementing the Configuration, Transformation, and Execution Layer

On the configuration layer we reverted to the SeMFIS platform, which offers a generic import and export interface for models in the ADOxml format. The transformation layer was realized using XSLT, and for the execution layer we used Stanford Protégé and the included APIs. In the following we will discuss the used technologies and platforms as well as the developed mechanisms and algorithms in more detail.

### 5.1. Technology and Platforms

The modeling language was implemented by extending the SeMFIS platform [10], which is based on metamodeling platform ADOxx [12]. The modeling languages for OWL ontologies as well as for frames ontologies are already part of SeMFIS. The introduced modeling language for SWRL references to constructs of the OWL modeling language. The attributes of the model elements can be accessed using a property window in SeMFIS. An excerpt of the property window of a class from the ontology model (see Figure 5b) is depicted in Figure 9. The entries of the listbox represent model hyperlinks to the class elements *c* and *d*.

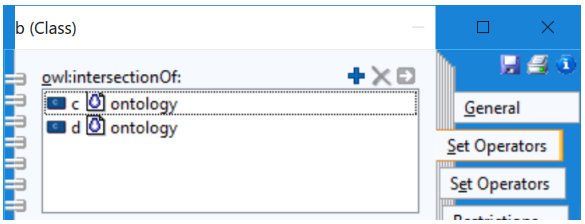



Fig. 9. Property Window in SeMFIS for Accessing Attributes of Model Elements - Example of Class Element *b* from the Ontology Model Depicted in Figure 5b

In SeMFIS, each model can be exported to a generic, standardized XML format called ADOxml[8], which is provided by the underlying metamodeling platform. ADOxml has a flat structure (limited hierarchy) and serves as input for the transformation layer.

The transformation layer has been realized with XSLT. It transforms ADOxml to OWL-XML and vice versa. The transformation layer has been integrated into the SeMFIS platform to ensure a seamless user experience. For the implementation, the XSLT transformation engine Saxon[9] was used. The resulting ontology together with the rules is stored in the OWL-XML syntax and can be processed in the execution layer. We used Stanford Protégé[10] for executing the generated ontology including the rules. Based on the transformation rules for the deserialization, also the import of existing rules in OWL-XML to the SeMFIS platform is supported.

### 5.2. Model Algorithms and Mechanisms

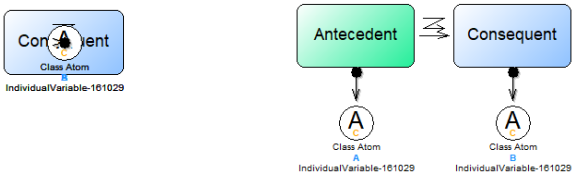
In addition to the import and export functionalities, we implemented further algorithms and mechanisms for the SWRL modeling language. In the following we focus on a layout and a validation algorithm.

**Layouting.** The OWL serializations do not contain any positioning information due to their lack of a graphical representation. To compensate for this, we used a layout algorithm that had been developed to ensure a proper positionining of model elements when importing SWRL rules [9, 35]. An example is shown in Figure 10, where a rule model is shown before and after the layout algorithm is applied.

[8]The schema is defined in https://www.adoxx.org/AdoScriptDoc/files/Message_Ports/Component_APIs/Documentation/XML_MODELS-js.html, accessed: 14.08.2017

[9]http://saxon.sourceforge.net/

[10]http://protege.stanford.edu/




(a) Model before Running the Layout Algorithm (b) Model after Running the Layout Algorithm

Fig. 10. Layout of Imported Models

The basic working of the algorithm is as follows. For every serialization that is imported, the algorithm first positions the *Antecedent* and the *Consequent* elements at the top of the model so that the edge of type *HasConsequent* is positioned between them. Next, the atoms connected to the antecedent are positioned in a row below the antecedent, followed by atoms that are connected both to the antecedent and the consequent. The atoms that are only connected to the consequent are placed at the end of the row. In addition, separate rows of elements are created below for the variables and constant terms.

In the ontology model, a separate row is created for all intances of a certain metaclass, the order is the following: Namespace, Class, Property, Instance, Predicate, AllDifferent and Package. The algorithm works in a responsive way, which means that if an entire row is being deleted, the objects below will "move up" one row if the layout algorithm is applied again.

**Validating.** A basic validation algorithm has also been implemented for the SWRL modeling language. Before a rule is exported, it is checked if e.g. an antecedent and a consequent exists and if they are connected. Further, it is checked if the atoms are connected to either an antecedent or a consequent. In cases in which such a possible invalidity is detected, the modeler is shown a warning.

## 6. Evaluation

For the evaluation of our approach, we took two directions. First, we evaluated our approach along the examples provided by the W3C in the SWRL proposal. These are used to illustrate the capabilities of SWRL and can therefore serve as examples that are easily accessible and well documented. Second, we applied our approach to a use case to assess how the approach would behave for an existing comprehensive rule set.

### 6.1. W3C Examples

From a general perspective, artifacts in design-oriented and engineering research can be evaluated using different approaches [57]. In the case of the introduced visual modeling language for SWRL, the most important aspect was to assess, whether the models created with it conform to the recommendation by W3C for SWRL and its extension introduced [7]. We evaluated the completeness by modeling the example rules introduced in the SWRL recommendation[11]. Each example was modeled using the introduced modeling method. Afterwards, we exported the SWRL model and manually inspected its serialization. The serialization was imported to Stanford Protégé in order find out if the widely used components such as the SWRLAPI[12] and the OWLAPI[13] are able to read the serialization. Finally, the SWRL ontology was exported using Protégé and re-imported to the extended SeMFIS platform. Table 2 shows that our approach supports also the other examples provided by the W3C.

We then modeled the SWRL rule presented in Example 5.1-3 of the SWRL recommendation[14] in Figure 12. Moreover, we added an anonymous class to the referenced OWL model in order to illustrate how our serialization/deserialization approach works for anonymous classes. The top left corner of the figure shows the SWRL model, which has model hyperlinks to the OWL model. These two models are serialized to OWL-XML. An excerpt of the serialized code is shown in Listing 5. This serialization can e.g. imported to Stanford Protégé, which offers a rule engine as well as a text-based editor for modifying and writing rules. Rules stored in a standardized syntax can be imported to our modeling toolkit as excerpts of the models depicted on the right upper corner show. The used layout algorithm determines the position of the modeling elements. This figure also reveals, that three anonymous classes are in the OWL ontology, while the initial model contains two anonymous classes. This is a result of the introduced multi-class strategy which we described before. Listing 5 shows the three axioms which result in an anonymous class: the two axioms contained in the *ObjectIntersectionOf* and the *ObjectIntersectionOf* axiom, which is contained in the *UnionOf* axiom in the second *EquivalentClass* axiom. In the use

[11] https://www.w3.org/Submission/SWRL: examples 5.1-1 - 5.1-6
[12] https://github.com/protegeproject/swrlapi
[13] https://github.com/owlcs/owlapi
[14] https://www.w3.org/Submission/SWRL/#5.1

| W3C Example | Rule | Supported by our Approach |
|---|---|---|
| Example 5.1-1 | hasParent(?x1, ?x2) ^ hasBrother(?x2, ?x3) -> hasUncle(?x1, ?x3) | ✓ |
| Example 5.1-2 | hasParent(?x1, ?x2) ^ hasSibling(?x2, ?x3) ^ hasSex(?x3, male) -> hasUncle(?x1, ?x3) | ✓ |
| Example 5.1-3 | Artist(?x) ^ Style(?y) ^ artistStyle(?x, ?y) ^ creator(?x, ?z) -> period(?z, ?y) | ✓ |
| Example 5.1-4 | *Excerpt:* LocationClass(?maploc) ^ psameLocation(?loc, ?maploc) ^ latitude(?loc, ?lat) ^ longitude(?loc, ?lon) -> latitude(?maploc, ?lat) ^ longitude(?maploc, ?lon) | ✓ |
| Example 5.1-5 | lengthInFeet(?instance,?feet) -> swrlb:multiply(?inches,?feet) ^ lengthIn-Inches(?instance,?inches) | ✓ |
| Example 5.1-6 | hasStatus(?customer, Gold) ^ hasTotalPurchase(?customer, ?total) ^ swrlb:greaterThanOrEqual(?total, 500) -> hasDiscount(?customer, 10) | ✓ |

Table 2
Evaluation of the Approach Along the Examples Provided by the W3C

case, we did not use an anonymous class in e.g. the class atom, as it is not supported by the SWRLtab of Stanford Protégé[15].

### 6.2. Use Case

In this section we describe how to use our approach for modifying an existing set of rules. Therefore we reverted to the SWRL rule set published within the Protégé project[16]. One of these rules is depicted on the top of Figure 12. We imported the OWL-XML serialization of this rule set with our application. The visualizations of the rules are depicted in the center of Figure 12. We removed other rules to improve the readability of the figure. In the extended SeMFIS modeling toolkit we added inter-alia an *individual-valued property* atom, termed *hasParent*, to a rule - the newly added atom is highlighted in Figure 12. After adding the atom, we serialized the model so that it can be processed by SWRL rule engines. We exported the model to OWL-XML and re-imported it to Protégé where we added individuals in order to infer and test the modified set of rules. Figure 13 shows the modified rule in the Protégé SWRLTab editor before it was processed by a rule engine. In this way the round-trip from an existing rule set to our visual modeling approach and back to another platform could be positively tested.

We plan further evaluations for assessing the performance of the introduced transformation rules. Additionally, usability tests are planned for testing the usability of the developed approach.

[15]https://github.com/protegeproject/swrlapi/wiki/SWRLAPISWRLSyntax

[16]http://swrl.stanford.edu/ontologies/examples/family.swrl.owl

**LISTING 5:** Excerpt of the Serialization of the Use Case

```
[...]
<owlx:EquivalentClasses >
    <owlx:Class IRI= A / >
    <owlx:ObjectIntersectionOf >
        <owlx:ObjectIntersectionOf >
            <owlx:Class IRI= B / >
            <owlx:Class IRI= C / >
        </owlx:ObjectIntersectionOf>
        <owlx:ObjectUnionOf >
            <owlx:Class IRI= Artist / >
            <owlx:Class IRI= Style / >
        </owlx:ObjectUnionOf>
    </owlx:ObjectIntersectionOf>
</owlx:EquivalentClasses>
<owlx:EquivalentClasses >
    <owlx:Class IRI= A / >
    <owlx:ObjectUnionOf >
        <owlx:ObjectIntersectionOf >
            <owlx:Class IRI= Artist / >
            <owlx:Class IRI= Style / >
        </owlx:ObjectIntersectionOf>
    </owlx:ObjectUnionOf>
</owlx:EquivalentClasses>
<owlx:DLSafeRule >
    <owlx:Body >
        <owlx:ClassAtom >
            <owlx:Class IRI= Artist / >
            <owlx:IndividualVariable IRI= / >
                x1
        < /owl:ClassAtom>
        [...]
    < /owl:Body>
    [...]
< /owl:DLSafeRule>
[...]
```

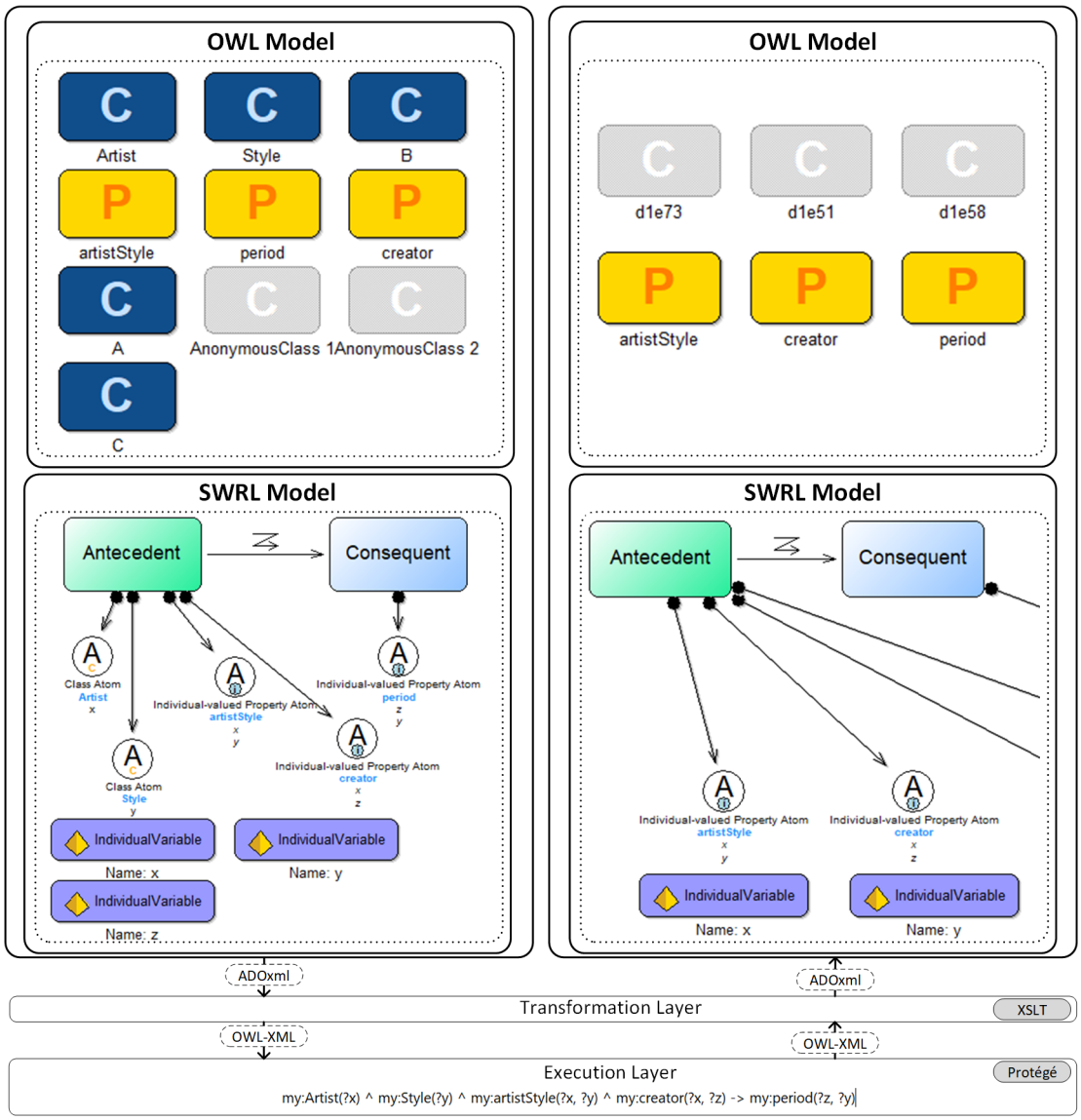



Fig. 11. Use Case - Export of Model, Processing and Import of Model

## 7. Discussion and Limitations

In industrial settings, the support of non-technical users for managing complex formalisms and formal systems is essential [9]. With the presented visual modeling language for SWRL, domain experts can use the predefined modeling elements and algorithms for composing rules. In this way, the modeling language can be considered as a substitution for the widely-used text-based syntax for SWRL rules. An initial empirical study, which attests the benefit of modeling rules, was introduced in [8]. It was found that the visual approach outperforms the text-based approach in all measured categories in both user groups - experts and novices. However, further empirical evaluation will be necessary due to the small sample size. The described visual approach is also useful for visualizing existing rules to domain experts which have to document, improve and inspect them.

Compared to the textual syntax, the introduced modeling language is able to re-use existing rule parts. The independence of our tool requires the usage of standardized serialization formats. Therefore, we implemented in addition to the validation and layout algo-

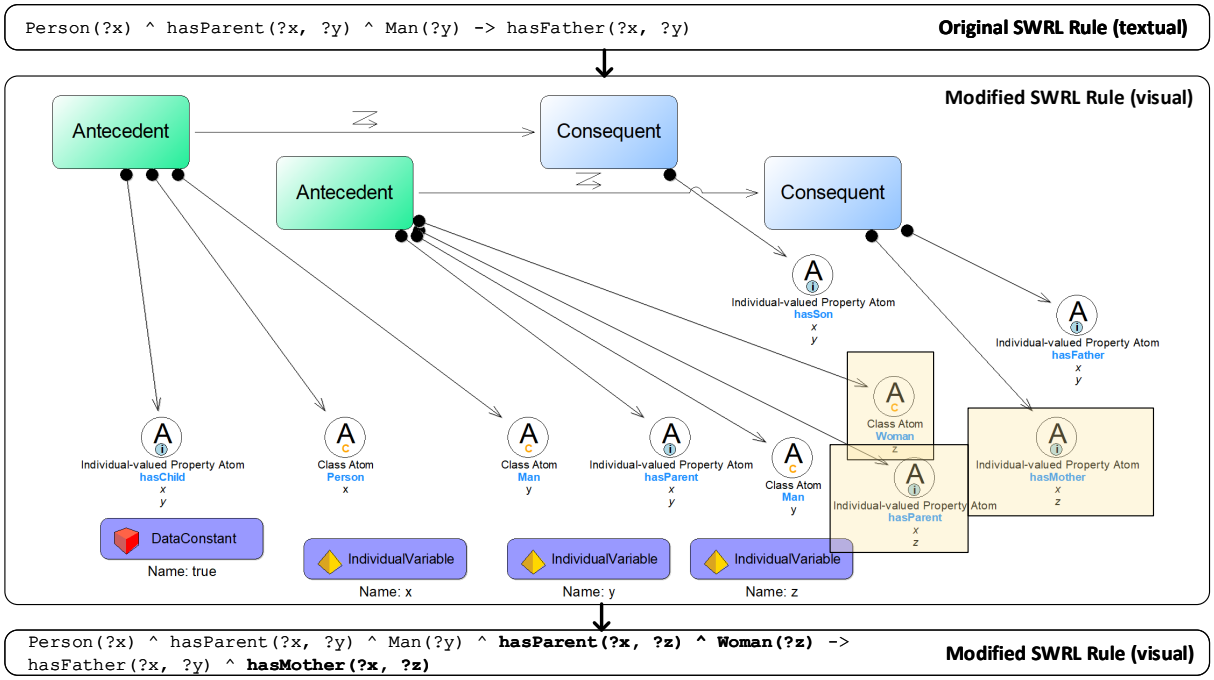



Fig. 12. Extended Visualization of the Rules (Excerpt)

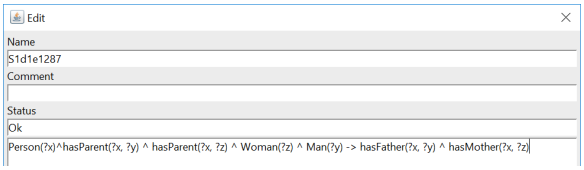



Fig. 13. Screenshot of a Rule in the Stanford Protégé Editor

rithms, import and export facilities for the standardized OWL-XML syntax. This enables compatibility with semantic web platforms such as Stanford Proégé and other industry applications.

Even if the visual modeling approach simplifies the creation of SWRL rules, the modelers have to understand the basic concepts of SWRL. For large rules, e.g. with thousands of atoms, advanced visualization techniques such as groupings will have to be introduced.

While we considered an inclusion of a SWRL engine to our modeling toolkit, we rather decided to support standardized SWRL syntaxes so that external rule engines can process them. This is because we see the modeling tool as a platform- and engine- independent application.

Currently, the support of annotations in the SWRL modeling language is limited. We will add this functionality within the next SeMFIS release.

In the current implementation, a visual ontology model has to be created or imported. While the introduced import function allows to import already existing ontologies, there are several use cases for which direct references to OWL serializations - not to the visual model - might be beneficial. For example in cases, in which the ontology is gathered from the web.

The current implementation focuses on the support of the OWL-XML syntax. Libraries such as OWLAPI or SWRLAPI can be used to convert OWL-XML serializations to other concrete syntaxes.

## 8. Conclusion and Further Research

In this paper we conceptualized a visual modeling language for SWRL rules with a special aim on supporting non-technical domain experts and business users. Instead of *programming* SWRL using a text-based editor such as used in the SWRLtab of Stanford Protégé, our modeling approach allows constructing SWRL rules using predefined modeling classes. Domain experts can use these predefined classes - by *drag and drop* and thus construct the rule. Validation algorithms support the users during the creation of the rules.

For visualizing and modifying already existing rules, our approach provides an import functionality. Additionally, the rule models can be exported in order to process the rules using rule engines. Therefore, we introduced generic transformation rules in the paper which can be adapted for syntaxes such as RDF-XML. We also discussed issues which occur during the transformations and strategies for handling them.

The introduced approach has been implemented by extending the SeMFIS platform. The transformation was realized with XSLT whereby we support the OWL-XML syntax for importing and exporting rules.

In our further research we plan to add support for further syntaxes such as RDF-XML. With a special aim on assisting business users we want to improve validation algorithms (e.g. by making suggestions for solving validity issues). We also consider providing mechanisms which allow direct references from SWRL atoms to OWL ontology concepts - currently OWL ontologies have to be imported as a model in order to reference their concepts. An empirical evlaution of the benefits of the introduced visual approach is necessary and will be a main pillar for further improvements.

## References


[1] R. Studer, V.R. Benjamins and D. Fensel, Knowledge Engineering: Principles and Methods, *Data Knowl. Eng.* **25**(1–2) (1998), 161–197.

[2] S. Stadtmueller, S. Speiser, A. Harth and R. Studer, Data-Fu: A Language and an Interpreter for Interaction with Read/Write Linked Data, in: *WWW Conference*, 2013, pp. 1225–1236.

[3] D. Feldkamp, K. Hinkelmann and B. Thönssen, KISS - Knowledge-Intensive Service Support for Agile Process Management, in: *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management*, Springer, 2007.

[4] T.A. Pham and N. Le Than, Checking the Compliance of Business Processes and Business Rules Using OWL 2 Ontology and SWRL, in: *AECIA Conference 2015*, Springer, 2016, pp. 11–20.

[5] M. O'Connor, H. Knublauch, S. Tu and M. Musen, Writing rules for the semantic web using SWRL and Jess, *Protégé With Rules Workshop, Madrid* (2005).

[6] C. Keßler, A RESTful SWRL Rule Editor, in: *Web Reasoning and Rule Systems - Fourth International Conference, RR 2010*, 2010, pp. 235–238.

[7] B. Glimm, M. Horridge, B. Parsia and P.F. Patel-Schneider, A Syntax for Rules in OWL 2, in: *Proceedings of the 5th International Workshop on OWL: Experiences and Directions*, 2009. http://ceur-ws.org/Vol-529/owled2009_submission_16.pdf.

[8] K. Skillen, L. Chen and W. Burns, VIPR: A Visual Interface Tool for Programming Semantic Web Rules, in: *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress*, IEEE, 2016, pp. 277–284.

[9] H. Fill, B. Pittl and G. Honegger, A Modeling Environment for Visual SWRL Rules Based on the SeMFIS Platform, in: *Designing the Digital Transformation - 12th International Conference DESRIST*, 2017, pp. 452–456.

[10] H. Fill, SeMFIS: A Flexible Engineering Platform for Semantic Annotations of Conceptual Models, *Semantic Web* **8**(5) (2017), 747–763.

[11] H.-G. Fill, On the Conceptualization of a Modeling Language for Semantic Model Annotations, in: *Advanced Information Systems Engineering Workshops, CAiSE 2011*, C. Salinesi and O. Pastor, eds, Springer, 2011, pp. 134–148.

[12] H. Fill and D. Karagiannis, On the Conceptualisation of Modelling Methods Using the ADOxx Meta Modelling Platform, *Enterprise Modelling and Information Systems Architectures* **8**(1) (2013), 4–25.

[13] D. Karagiannis and H. Kühn, Metamodelling Platforms, in: *E-Commerce and Web Technologies, Third International Conference, EC-Web*, K. Bauknecht, A. Min Tjoa and G. Quirchmayr, eds, 2002, p. 182.

[14] D. Bork and H.-G. Fill, Formal aspects of enterprise modeling methods: a comparison framework, in: *2014 47th Hawaii International Conference on System Sciences*, IEEE, 2014, pp. 3400–3409.

[15] N. Visic, H.-G. Fill, R.A. Buchmann and D. Karagiannis, A Domain-specific Language for Modeling Method Definition: from Requirements to Grammar, in: *Ninth International Conference on Research Challenges in Information Science*, IEEE, 2015.

[16] B. Schätz, Formalization and rule-based transformation of EMF Ecore-based models, in: *International Conference on Software Language Engineering*, Springer, 2008, pp. 227–244.

[17] H. Fill, T. Redmond and D. Karagiannis, FDMM: A Formalism for Describing ADOxx Meta Models and Models, in: *ICEIS 2012 - Proceedings of the 14th International Conference on Enterprise Information Systems, Volume 3*, L. Maciaszek, A. Cuzzocrea and J. Cordeiro, eds, 2012, pp. 133–144.

[18] H.-G. Fill, S. Hickl, D. Karagiannis, A. Oberweis and A. Schoknecht, A Formal Specification of the Horus Modeling Language Using FDMM, in: *International Conference on Business Informatics 2013*, AIS, 2013.

[19] F. Johannsen and H.-G. Fill, Meta Modeling for Business Process Improvement, *Business & Information Systems Engineering* **59**(4) (2017), 251–275.

[20] T. Glässner, F. Heumann, L. Keßler, F. Härer, A. Steffan and H.-G. Fill, Experiences from the Implementation of a Structured-Entity-Relationship Modeling Method in a Student Project, in: *Proceedings of the 1st International Workshop on Practicing Open Enterprise Modeling within OMiLAB (PrOse 2017)*, D. Bork, D. Karagiannis and J. Vanthienen, eds, CEUR, 2017.

[21] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean et al., SWRL: A semantic web rule language combining OWL and RuleML, *W3C Member submis-*

*sion* **21** (2004), 79, Accessed on 2017-07-10. https://www.w3.org/Submission/SWRL/.

[22] T. Eiter, G. Ianni, T. Krennwallner and A. Polleres, Rules and Ontologies for the Semantic Web, in: *Reasoning Web, 4th International Summer School 2008*, 2008, pp. 1–53.

[23] D.L. McGuinness, F. Van Harmelen et al., OWL web ontology language overview, *W3C recommendation* **10**(10) (2004), 2004, Accessed on 2017-07-10. https://www.w3.org/TR/owl-features/.

[24] B. Motik, P.F. Patel-Schneider and I. Horrocks, OWL 2 web ontology language: Structural specification and functional-style syntax, *W3C recommendation* **11** (2009), Accessed on 2017-07-10. https://www.w3.org/TR/owl2-syntax/.

[25] B. Glimm, I. Horrocks, B. Motik, G. Stoilos and Z. Wang, HermiT: An OWL 2 Reasoner, *J. Autom. Reasoning* **53**(3) (2014), 245–269.

[26] M.A. Musen, The protégé project: a look back and a look forward, *AI Matters* **1**(4) (2015), 4–12.

[27] M. Horridge, OWL Syntaxes, *Ontogenesis* (2010), Accessed on 2017-07-10. http://ontogenesis.knowledgeblog.org/88.

[28] S. Bechhofer, B.C. Grau, A. Fokoue, R. Hoekstra and B. Parsia, OWL 2 Web Ontology Language XML Serialization, *W3C recommendation* **27**(65) (2009), 159, Accessed on 2017-07-10. https://www.w3.org/TR/2009/REC-owl2-xml-serialization-20091027/#Appendix:_The_Derivation_from_the_Functional_Syntax_.28Informative.29.

[29] F. Khan, A. Bellandi, F. Frontini and M. Monachini, Using SWRL Rules to Model Noun Behaviour in Italian, in: *First International Conference on Language, Data, and Knowledge*, 2017, pp. 134–142.

[30] C.S. Namahoot, S. Sivilai and M. Brückner, An Ingredient Selection System for Patients Using SWRL Rules Optimization and Food Ontology, in: *International Conference on Cooperative Design, Visualization, and Engineering*, 2016, pp. 163–171.

[31] E. Cardillo, M.T. Chiaravalloti, C. Eccher, E. Pasceri, V.D. Mea, L. Frattura and R. Guarasci, Towards a Rule-based Support System for the Coding of Health Conditions in the Patient Summary, in: *Proceedings of International Workshop on Biomedical Data Mining, Modeling, and Semantic Integration*, 2015.

[32] V. Lombardo, C. Battaglino, A. Pizzo, R. Damiano and A. Lieto, Coupling Conceptual Modeling and Rules for the Annotation of Dramatic Media, *Semantic Web* **6**(5) (2015), 503–534.

[33] P. Bonte, F. Ongenae, J. Schaballie, D. Arndt, P. Leroux, R. Verborgh, E. Mannens, F.D. Turck and R. Wauters, Semantic Intelligence for Real-time Automated Media Production, in: *Proceedings of the ISWC 2015 Posters & Demonstrations Track*, 2015.

[34] E.F. Aminu, O.N. Oyelade and I.S. Shehu, Rule Based Communication Protocol between Social Networks using Semantic Web Rule Language (SWRL), *International Journal of Modern Education and Computer Science* **8**(2) (2016), 22.

[35] B. Pittl, H. Fill and G. Honegger, Enabling Risk-Aware Enterprise Modeling using Semantic Annotations and Visual Rules, in: *European Conference on Information Systems*, AIS, 2017.

[36] J.L.R. Moreira, T.P. Sales, J. Guerson, B.F.B. Braga, F. Brasileiro and V. Sobral, Menthor Editor: An Ontology-Driven Conceptual Modeling Platform, in: *Proceedings of the Joint Ontology Workshops 2016*, 2016.

[37] J. Bak, M. Nowak and C. Jedrzejek, RuQAR: Reasoning Framework for OWL 2 RL Ontologies, in: *The Semantic Web: ESWC 2014 Satellite Events - Revised Selected Papers*, 2014, pp. 195–198.

[38] K. Kim, H. Kim, S. Kim and J. Jung, i-RM: An intelligent risk management framework for context-aware ubiquitous cold chain logistics, *Expert Syst. Appl.* **46** (2016), 463–473.

[39] T.W. Wlodarczyk, C. Rong, M.J. O'Connor and M.A. Musen, SWRL-F: a fuzzy logic extension of the semantic web rule language, in: *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, 2011, p. 39.

[40] E. Jajaga and L. Ahmedi, C-SWRL: SWRL for Reasoning over Stream Data, in: *11th IEEE International Conference on Semantic Computing*, 2017, pp. 395–400.

[41] A. Boujelben, T. Chaari and I. Amous, Towards Better SWRL Rules Dependency Extraction, in: *16th International Conference on Intelligent Systems Design and Applications*, 2016, pp. 781–790.

[42] S. Brockmans, P. Haase and H. Stuckenschmidt, Formalism-Independent specification of ontology mappings–a metamodeling approach, *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE* (2006), 901–908.

[43] A. Leutgeb, W. Utz, R. Woitsch and H.-G. Fill, Adaptive Processes in E-Government - A Field Report about Semantic-based Approaches from the EU-Project FIT, in: *ICEIS'2007*, INSTICC, 2007, pp. 264–269.

[44] A. Leutgeb, *The business rules method : a modeling method for adaptive processes - Master Thesis*, University of Vienna, Wien, 2007.

[45] J. Bak, M. Nowak and C. Jedrzejek, Graph-based Editor for SWRL Rule Bases, in: *Joint Proceedings of the 7th International Rule Challenge*, 2013.

[46] S. Hassanpour, M.J. O'Connor and A.K. Das, A Rule Management and Elicitation Tool for SWRL Rule Bases, in: *Proceedings of the 3rd International RuleML-2009 Challenge*, 2009.

[47] M.K. Sarker, D. Carral, A.A. Krisnadhi and P. Hitzler, Modeling OWL with Rules: The ROWL Protege Plugin, in: *Proceedings of the ISWC 2016 Posters & Demonstrations Track*, 2016.

[48] D. Harel and B. Rumpe, Modeling Languages: Syntax, Semantics and All That Stu (2000). http://www4.in.tum.de/publ/papers/HR00.pdf.

[49] H.-G. Fill and P. Burzynski, Integrating Ontology Models and Conceptual Models using a Meta Modeling Approach, in: *11th International Protégé Conference*, 2009.

[50] H.-G. Fill, Design of Semantic Information Systems using a Model-based Approach, in: *AAAI Spring Symposium*, AAAI, 2009.

[51] D. Moody, The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering, *IEEE Transactions on Software Engineering* **35**(6) (2009), 756–779.

[52] ArchiMate, ArchiMate, *The Open Group* (2016), Accessed on 2017-07-10. https://www2.opengroup.org/ogsys/catalog/C162.

[53] UML, Unified Modeling Langague Specification (UML), *OMG* (2015), Accessed on 2017-07-10. http://www.omg.org/spec/UML/2.5/PDF/.

[54] BPMN, Business Process Model and Notation (BPMN), *OMG* (2013), Accessed on 2017-07-10. http://www.omg.org/spec/BPMN/2.0.2/PDF/.

[55] H.-G. Fill, *Visualisation for Semantic Information Systems*, Gabler, 2009.
[56] D. Harel and B. Rumpe, Meaningful Modeling: What's the Semantics of "Semantics"?, *IEEE Computer* (2004), 64–72.
[57] K. Peffers, T. Tuunanen, M.A. Rothenberger and S. Chatterjee, A Design Science Research Methodology for Information Systems Research, *JMIS* **24**(3) (2007), 45–77.

## Appendix A. SWRL Modeling Language in FDMM

In the following we list the details of the SWRL modeling language as expressed in the FDMM formalism [17]. The visual modeling language for the SWRL consists of a single model type $\mathbf{MT}_{SWRL}$. This model type has a set of object types $\mathbf{O}^T_{SWRL}$ which have a set of attributes $\mathbf{A}_{SWRL}$ which have in turn data types $\mathbf{D}^T_{SWRL}$. The object types used in the model type are described in the following equation. In FDMM connectors such as *HasAtom* and *HasConsequent* are also considered as object types. We focused on the core elements and therefore we neglected the meta-classes *MetaInfo* and *Annotation*.

$$\begin{aligned}\mathbf{O}^T_{SWRL} =\{&Consequent, Antecedent,\\ &RulePart, Atom,\\ &ClassAtom, IndividualPropertyAtom,\\ &DataValuedPropertyAtom, DataRangeAtom,\\ &DifferentIndividualsAtom, BuiltInAtom,\\ &SameIndividualsAtom, DataRangeAtomList,\\ &IndividualVariable, IndividualConstant,\\ &DataVariable, DataConstant, BuildInTerm,\\ &HasAtom, HasConsequent\}\end{aligned} \tag{1}$$

All attributes used in the object types are part of the set $\mathbf{A}_{SWRL}$. In the following, only the most important attributes are given.

$$\begin{aligned}\mathbf{A}_{SWRL} =\{&Name, RepresentsClass,\\ &RepresentsProperty, Datatype,\\ &Individualvalue1, Individualvalue2,\\ &Classname, Propertyname, Datavalue,\\ &Individuals, Literallist, BuiltInName,\\ &HasConsequent\_from, HasConsequent\_to,\\ &HasAtom\_from, HasAtom\_to,\\ &BuildInTerms, Value, RepresentsIndividual\}\end{aligned} \tag{2}$$

The datatypes of the attributes are summarized in the set $\mathbf{D}^T_{SWRL}$. $\text{Enum}_{\text{builtIn}}$ represents a enumeration list. Due to the space constraints we do not list all its values.

$$\begin{aligned}\mathbf{D}^T_{SWRL} =\{&String, IndividualVariable, OWLClass,\\ &Enum_{\text{builtIn}} = \{greaterThan, smallerThan, ...\\ &\}, OWLProperty, DataConstant, Data,\\ &Enum_{\text{datatype}} = \{string, integer, ...\},\\ &Consequent, Atom, IndividualConstant,\\ &BuildInTerm, DataVariable, Individual\}\end{aligned} \tag{3}$$

The ordering of the object types is described in the following:

$$\begin{aligned} Consequent &\preceq RulePart\\ Antecedent &\preceq RulePart\\ ClassAtom &\preceq Atom\\ IndividualVariable &\preceq Individual\\ IndividualConstant &\preceq Individual\\ DataConstant &\preceq Data\\ DataVariable &\preceq Data\\ IndividualPropertyAtom &\preceq Atom\\ DataValuedPropertyAtom &\preceq Atom\\ DataRangeAtom &\preceq Atom\\ DifferntIndividualAtom &\preceq Atom\\ SameIndividualAtom &\preceq Atom\\ BuiltInAtom &\preceq Atom\\ DataRangeAtom &\preceq Atom \end{aligned} \tag{4}$$

In the following, the domain functions are used to establish a mapping between the attributes and the object types:

$$\begin{aligned}\text{domain}(Name) =\{&Atom, RulePart,\\ &IndividualConstant,\\ &IndividualVariable,\\ &DataVariable,\\ &DataConstant, BuiltInTerm\},\\ \text{domain}(RepresentsClass) =\{&ClassAtom\},\\ \text{domain}(RepresentsProperty) =\{&IndividualPropertyAtom,\\ &DataValuedPropertyAtom\},\\ \text{domain}(Datatype) =\{&DataConstant,\end{aligned}$$

$$
\begin{aligned}
&\textit{DataRangeAtom}\},\\
\text{domain}(\textit{Individualvalue1}) =&\{\textit{ClassAtom},\\
&\textit{IndividualPropertyAtom},\\
&\textit{DataValuedPropertyAtom}\},\\
\text{domain}(\textit{Individualvalue2}) =&\{\textit{IndividualPropertyAtom}\},\\
\text{domain}(\textit{Classname}) =&\{\textit{ClassAtom}\},\\
\text{domain}(\textit{Propertyname}) =&\{\textit{IndividualPropertyAtom},\\
&\textit{DataValuedPropertyAtom}\},\\
\text{domain}(\textit{Datavalue}) =&\{\textit{DataValuedPropertyAtom},\\
&\textit{DataRangeAtom},\\
&\textit{DataRangeAtomList},\\
&\textit{BuiltInTerm}\},\\
\text{domain}(\textit{Individuals}) =&\{\textit{DifferntIndividualAtom},\\
&\textit{SameIndividualAtom}\},\\
\text{domain}(\textit{Literallist}) =&\{\textit{DataRangeAtomList}\},\\
\text{domain}(\textit{BuiltInName}) =&\{\textit{BuildInAtom}\},\\
\text{domain}(\textit{BuildInTerms}) =&\{\textit{BuildInAtom}\},\\
\text{domain}(\textit{HasConsequent\_to}) =&\{\textit{HasConsequent}\},\\
\text{domain}(\textit{HasConsequent\_from}) =&\{\textit{HasConsequent}\},\\
\text{domain}(\textit{HasAtom\_from}) =&\{\textit{HasAtom}\},\\
\text{domain}(\textit{HasAtom\_to}) =&\{\textit{HasAtom}\},\\
\text{domain}(\textit{Value}) =&\{\textit{DataConstant}\}. \qquad (5)
\end{aligned}
$$

The range function assigns datatypes to attributes used in object types. The range functions are listed in the following:

$$
\begin{aligned}
\text{range}(\textit{Name}) =&\{\textit{String}\},\\
\text{range}(\textit{RepresentsClass}) =&\{\textit{OWLClass}\},\\
\text{range}(\textit{RepresentsProperty}) =&\{\textit{OWLProperty}\},\\
\text{range}(\textit{Datatype}) =&\{\textit{Enum}_{\text{datatype}}\},\\
\text{range}(\textit{Individualvalue1}) =&\{\textit{IndividualConstant},\\
&\textit{IndividualVariable}\},\\
\text{range}(\textit{Individualvalue2}) =&\{\textit{IndividualConstant},\\
&\textit{IndividualVariable}\},\\
\text{range}(\textit{Classname}) =&\{\textit{String}\},\\
\text{range}(\textit{Propertyname}) =&\{\textit{String}\},\\
\text{range}(\textit{Datavalue}) =&\{\textit{DataConstant},\\
&\textit{DataVariable}\},\\
\text{range}(\textit{Individuals}) =&\{\textit{IndividalConstant},\\
&\textit{IndividualVariable}\},\\
\text{range}(\textit{Literallist}) =&\{\textit{DataConstant}\},\\
\text{range}(\textit{BuiltInName}) =&\{\textit{String}\},\\
\text{range}(\textit{BuildInTerms}) =&\{\textit{BuildInTerm}\},\\
\text{range}(\textit{HasConsequent\_to}) =&\{\textit{Consequent}\},\\
\text{range}(\textit{HasConsequent\_from}) =&\{\textit{Antecedent}\},\\
\text{range}(\textit{HasAtom\_from}) =&\{\textit{Antecedent}\},\\
\text{range}(\textit{HasAtom\_to}) =&\{\textit{Consequent}\},\\
\text{range}(\textit{Value}) =&\{\textit{String}\}. \qquad (6)
\end{aligned}
$$

In FDMM the cardinality function - abbreviated with *card* - defines how many attribute values a object type can have. In our modeling type all attributes have at most one value except the attributes *Individuals, Literallist* and *BuildInTerms* which can have an infinite number of parameters. Therefore, we do not show the cardinality functions in this paper.

In the following we describe an excerpt of the visual OWL model using the FDMM which we require for formalizing the model transformation.

$$
\begin{aligned}
\mathbf{O}^{T}_{OWL} =&\{\textit{OWLClass}, \textit{OWLProperty}\}\\
\mathbf{A}_{OWL} =&\{\textit{Name}, \textit{IntersectionOf}, \textit{UnionOf},\\
&\textit{IsAnonymous...}\}\\
\mathbf{D}^{T}_{OWL} =&\{\textit{String}, \textit{OWLClass}, ...\} \qquad (7)
\end{aligned}
$$

The domain functions are then described in the following.

$$
\begin{aligned}
\text{domain}(\textit{Name}) =&\{\textit{OWLClass}, \textit{OWLProperty}\},\\
\text{domain}(\textit{IntersectionOf}) =&\{\textit{OWLClass}\},\\
\text{domain}(\textit{UnionOf}) =&\{\textit{OWLClass}\},\\
\text{domain}(\textit{IsAnonymous}) =&\{\textit{OWLClass}\} \qquad (8)
\end{aligned}
$$

The corresponding range functions are:

$$
\begin{aligned}
\text{range}(\textit{Name}) =&\{\textit{String}\},\\
\text{range}(\textit{IntersectionOf}) =&\{\textit{OWLClass}\},\\
\text{range}(\textit{UnionOf}) =&\{\textit{OWLClass}\},\\
\text{range}(\textit{IsAnonymous}) =&\{\textit{Boolean}\}. \qquad (9)
\end{aligned}
$$

The cardinality functions are not formally described here. The attributes *Name* and *IsAnonymous* can have at most one value while the attributes *IntersectionOf* and *unionOf* can have multiple values.

## Appendix B. SWRL Serialization in FDMM

For a clear description of the transformation from the model to the serialization and vice versa we mapped the functional syntax described in [7] - which references to OWL 2 elements described in [24] - to the FDMM. The functional syntax is described using production rules and the mapping to FDMM was done as follows: (i) For each terminal symbol an object type in the FDMM was created except for the single brackets which are used in the functional syntax for grouping. An excerpt of a production rule used in [7] and [28] is shown in the following:

**DLSafeRule** ::= [...] 'Body' '(' {**Atom**} ')'
'Head' '(' {**Atom**} ')'
**Atom**::='ClassAtom' '(' **ClassExpression IArg** ')'
| 'DataRangeAtom' '(' **DataRange DArg** ')'
| 'SameIndividualAtom' '(' **IArg IArg** ')'
**IArg**::= [...] 'Variable' '(' **IRI** ')'
**IRI** ::= **fullIRI** | **abbreviatedIRI**
**fullIRI** ::= an IRI as defined in [RFC3987], enclosed in a pair of < (U+3C) and > (U+3E) characters
**abbreviatedIRI** ::= a finite sequence of characters matching the PNAME_LN production of [SPARQL]
[...]

For example, the FDMM object types *Body*, *Head*, *ClassAtom*, *DataRangeAtom*, *SameIndividualAtom* and *Variable* are created for the corresponding terminals. (ii) For all terminals a corresponding attribute in the FDMM is created for each non-terminal it contains using the prefix *Has*. So for the previous example, the object type *Body* and *Head* get the attribute *HasAtom*. The object type *ClassAtom* gets two attributes *HasClassExpression* and *HasIArg*. Similarly, for the object type *DataRangeAtom* the attributes *HasDataRange* and *HasDArg* are created while the object type *SameIndividualAtom* gets the attribute *HasIArg*. For the object type *Variable* the attribute *HasIRI* is created. (iii) The range of the attributes are the object types created within the non-terminal for which the attribute was created. For example, the range of the previous used attribute *HasIArg* consists of the object type *Variable*. Some contained production rules do not result into object types because the production rule refers to simple datatypes which are used instead as range for attributes in the FDMM. (iv) The non-terminals which are not grouped by a terminal are replaced in the mapping process to the FDMM by their content. So production rules which do not consist of terminals do not result into an object type in the FDMM. So e.g. the range of the attribute *HasIRI* is a string as the production rule *IRI* refers finally to string representing a IRI[17].

The following description using the FDMM starts with the production rule *DLSafeRule*. To stay compliant with the SWRL modeling language we did not consider annotations as well as all OWL constructs e.g. we simplified production rules of the OWL *Literal*.

The serialization of the rules is described using the model type $\mathbf{MT}_{\overline{SWRL}}$. The model type has a set of object types $\mathbf{O}^{T}_{\overline{SWRL}}$ which have a set of attributes $\mathbf{A}_{\overline{SWRL}}$ which have in turn data types $\mathbf{D}^{T}_{\overline{SWRL}}$. The object types used in the model type are described in the following equation.

$$\begin{aligned}\mathbf{O}^{T}_{\overline{SWRL}} = \{&Head, Body,\\ &ClassAtom, ObjectPropertyAtom,\\ &DataPropertyAtom, DataRangeAtom,\\ &DifferentIndividualsAtom, BuiltInAtom,\\ &SameIndividualsAtom, Variable, Literal\\ &ObjectItersectionOf, DataOneOf,\\ &ObjectUnionOf, ...\}\end{aligned} \tag{10}$$

Unlike in the model where model instances exist independent of other instances, in the serialization model atoms can only occur within a single head or body. Similarly, variables and class expressions such as instances of the object type *ObjectInstersectionOf* can only occur within a single atom leading to the following conditions whereby $k$ is a placeholder for the atoms:

$$\begin{aligned}&\forall k \in \{ClassAtom, ...\} \forall i \in \mu_{\mathbf{o}}(k, \mathbf{MT}_{\overline{SWRL}}) :\\ &\exists j \in \{\mu_{\mathbf{o}}(Head, \mathbf{MT}_{\overline{SWRL}}) \cup \mu_{\mathbf{o}}(Body,\\ &\mathbf{MT}_{\overline{SWRL}})\} | i \in j.HasAtom\\ &\wedge \nexists m \in \{\mu_{\mathbf{o}}(Head, \mathbf{MT}_{\overline{SWRL}}) \cup \mu_{\mathbf{o}}(Body,\\ &\mathbf{MT}_{\overline{SWRL}})\} | i \in m.HasAtom\\ &\wedge m \neq j\end{aligned} \tag{11}$$

[17] a distinction between the different string content is not done in the given transformation

$$
\begin{aligned}
&\forall g \in \{Variable, ObjectUnionOf, ...\} \forall i \in \\
&\mu_{\mathbf{o}}(g, \mathbf{MT}_{\overline{SWRL}}) : \\
&\exists j \in \{\mu_{\mathbf{o}}(ClassAtom, \mathbf{MT}_{\overline{SWRL}}) \cup \mu_{\mathbf{o}}( \\
&ObjectPropertyAtom, \mathbf{MT}_{\overline{SWRL}}) \cup ..\} \\
&|i \in \{j.HasIArg \cup j.HasDArg...\} \\
&\wedge \nexists m \in \{\mu_{\mathbf{o}}(ClassAtom, \mathbf{MT}_{\overline{SWRL}}) \cup \mu_{\mathbf{o}}( \\
&ObjectPropertyAtom, \mathbf{MT}_{\overline{SWRL}}) \cup ..\} \\
&|i \in \{m.HasIArg \cup m.HasDArg...\} \\
&\wedge m \neq j
\end{aligned} \tag{12}
$$

All attributes used in the object types are part of the set $\mathbf{A}_{\overline{SWRL}}$. In the following, only the most important attributes are described.

$$
\begin{aligned}
\mathbf{A}_{\overline{SWRL}} = \{&HasAtom, HasClassExpression, \\
&HasDataRange, HasIArg, HasDArg \\
&HasIRI, HasDataProperty, \\
&HasClassExpression, HasValue, \\
&HasDatatype, HasObjectProperty\}
\end{aligned} \tag{13}
$$

The datatypes of the attributes are summarized in the set $\mathbf{D}^{T}_{\overline{SWRL}}$.

$$
\begin{aligned}
\mathbf{D}^{T}_{\overline{SWRL}} = \{&String, ClassDescriptionExpression \\
&Variable, Literal, \\
&ClassAtom, DataRangeAtom, \\
&ObjectPropertyAtom, DataPropertyAtom, \\
&SameIndividualAtom, BuiltInAtom, \\
&DifferentIndividualsAtom\}
\end{aligned} \tag{14}
$$

In the following, the domain functions are used to establish a mapping between the attributes and the object types:

$$
\begin{aligned}
\text{domain}(HasAtom) &= \{Body, Head\}, \\
\text{domain}(HasClassExpression) &= \{ClassAtom, \\
&\quad ObjectIntersectionOf, \\
&\quad ObjectUnionOf\}, \\
\text{domain}(HasDataRange) &= \{DataRangeAtom\}, \\
\text{domain}(HasIArg) &= \{ClassAtom, \\
&\quad ObjectPropertyAtom, \\
&\quad DataPropertyAtom, \\
&\quad BuiltInAtom, \\
&\quad SameIndividualAtom, \\
&\quad DifferentIndividualsAtom\}, \\
\text{domain}(HasDArg) &= \{DataRangeAtom, \\
&\quad DataPropertyAtom, \\
&\quad BuiltInAtom, DataOneOf\}, \\
\text{domain}(HasIRI) &= \{Variable\}, \\
\text{domain}(HasDataProperty) &= \{DataPropertyAtom\}, \\
\text{domain}(HasValue) &= \{Literal\}, \\
\text{domain}(HasDatatype) &= \{Literal\}, \\
\text{domain}(HasObjectProperty) &= \{ObjectPropertyAtom\}.
\end{aligned} \tag{15}
$$

The range function assigns datatypes to attributes used in object types. The range functions are listed in the following:

$$
\begin{aligned}
\text{range}(HasAtom) &= \{ClassAtom, DataRangeAtom, \\
&\quad ObjectPropertyAtom, \\
&\quad DataPropertyAtom, \\
&\quad BuiltInAtom, \\
&\quad SameIndividualAtom, \\
&\quad DifferentIndividualsAtom\}, \\
\text{range}(HasClassExpression) &= \{ObjectIntersectionOf, \\
&\quad ObjectUnionOf, \\
&\quad String\}, \\
\text{range}(HasDataRange) &= \{String, DataOneOf\}, \\
\text{range}(HasIArg) &= \{Variable\}, \\
\text{range}(HasDArg) &= \{Variable, Literal\}, \\
\text{range}(HasIRI) &= \{String\}, \\
\text{range}(HasDataProperty) &= \{String\}, \\
\text{range}(HasValue) &= \{String\}, \\
\text{range}(HasDatatype) &= \{String\}, \\
\text{range}(HasObjectProperty) &= \{String\}.
\end{aligned} \tag{16}
$$

To save space we did not describe the cardinality function here. All attributes except *HasIRI, HasDatatype* and *HasValue* have an infinite number of values.