Sequential Presentation of Method for Integration the OWL DL and SWRL Using Protégé-owl API

Khala K.A.
International Research and Training Centre of Information Technologies and Systems
National Academy of Sciences and Ministry of Education of Ukraine,
Kyiv, Ukraine
Email: cecerongreat@ukr.net

***Abstract*—This paper is devoted to the description of a method of integration of OWL DL and rules with sequential presentation. In article need of similar integration and semantics of OWL DL are analyzed; the method on integration of OWL and rules with use of language of the rules SWRL is provided.**

Keywords—OWL; Markup Language; SWRL; case based reasoning; reasoned; knowledge acquisition system; Protege ontology editor

I. Introduction

In recent years, many formal ontologies were offered as the solution of problems of the description for difficult spheres of knowledge. Well thought over ontologies possess a row of the positive moments, including:

1) an opportunity to define controlled dictionaries of terms;
2) ability to inherit and expand the existing conditions;
3) an opportunity to declare correlations between the existing conditions;
4) an opportunity to add the new relations, on the basis of reasonings according to the existing terms.

By means of the technologies known under the general name Semantic Web, in particular the OWL [1] language, researchers can spread and be divided ontologies by all scientific community. Though there is a row of high-quality ontologies, scientists are far from implementation of all advantages of their use, and still there are opportunities for significant progress in this area, especially in application of the formal reasonings.

The unique force of the formal ontologies in the field of representation of knowledge is their ability to be accented on logical arguments. Such reasons are carried out with use description logicians (DL), forms of the logic developed for reasons about objects as separately and about classes of objects. The software under the name the moralist (Hermit, Pellet or Fact++) uses rules DL for execution of specific operations over knowledge bases [2]. The most important of them is:

1) coherence check: combining of ontological model with rules DL;
2) check of an consistency: ability for the described classes to be implemented by real copies;
3) classification: extension of the relations between objects which were brought out of the relations in an explicit form.

The OWL language offers rich property set, but apart from a set of relational properties of varieties of the OWL languages, they doesn't envelop all range of indicative opportunities for the relations of objects which can be constructed.

This article is devoted to the description of a method of integration of OWL DL and rules. For the solution of an objective, first of all, it is necessary to consider a question of feasibility of similar integration, then in details to analyze features of semantics of OWL DL. In the inference the method of integration of OWL and rules, SWRL (Semantic Web Rule Language) will be provided.

Frequent to model many processes in knowledge domain better with use of declarative approach and rules which results in interest in the systems based on rules. However, a possibility of interaction among a set of the existing systems based on rules limited. The SWRL technology [3] became the first step as based on the combining OWL with the rules Markup Language [4]. The combining OWL and SWRL gives opportunities of carrying out a logical output outside the opportunities of classification which are built in the description the logician, realized by OWL.

II. Use of a Combination of OWL DL and Rules

There are several reasons, for a choice of the OWL DL language as a formalism of model (legal) knowledge. First a key role is played by interchangeability and solubility. Secondly, among OWL family of languages, OWL DL is the most indicative dialect which remains solvable.

However there is one lack of OWL DL – limitation of its expressiveness, therefore, it needs to be expanded. One of methods consists in its extension using rules. The rule is a formula of a look:

$$\varphi_1 \wedge ... \wedge \varphi_k \rightarrow \psi$$

where $\varphi_1 \wedge ... \wedge \varphi_k \rightarrow \psi$ is a body, and ψ according to governed headed . The sense of the rule is that "every time when the body of the rule is true, the head will accept value truly too".

The choice governed as extension for OWL isn't accidental. First, integration of OWL (DL) into rules gives so indicative language as logic of first order (FOL – First Order Logic). Secondly, OWL DL well is suitable for expression of taxonomical, terminological or encyclopedic knowledge while rules can

express configurations of concepts and properties which can't be reduced to taxonomical classification, and it is necessary to express these configurations which OWL DL can't express [1,5].

For example, we will consider (informally expressed) the rule:

If the judge condemns the innocent,
then he is unfair to this person

Which conversion to FOL looks as follows:

$$\forall x, y : Judge(x) \wedge condemn(x, y) \wedge Innocent(y) \rightarrow unjust(x, y)$$

The example contains a ratio between concepts and properties which are necessary, but which OWL DL isn't capable to express. The intuitive reason is rather simple: to clarify that someone is a judge, the innocent, etc. (atoms in a rule body), is a subject of taxonomical knowledge and reasonings.

From the formal point of view, the shortcoming is caused by inability of OWL to cope with variables. Because in the rule variables are higher x, at are transferred from a body to the head, and OWL DL isn't able to cope with such transmission. Unlike it, the rule, such, as

If someone made action which isn't authorized,
then he made violation

Which are represented to FOL as:

$$\forall x, y : Person(x) \wedge Action(y) \wedge Disallowed(y) \rightarrow commit(x, violation)$$

It can be expressed in OWL as:

$$Person \sqcap \exists perform.Action \sqcap DisallowedPerson \sqcap \exists.commit \sqcap Violation.$$

Let's pay attention that in the last example only one variable is separated between a body and the head of this rule. Thus, it is possible to make two preliminary remarks:

- if at least one variable is separated between a body and the head of the rule, then such rule OWL DL represented;
- if more than one variable are the general, then the rule can't be OWL DL represented [6].

Thus, it was shown that there are rules which can't be expressed in OWL DL. From here need in a research of how to integrate OWL and rules follows [7].

III. Syntax and Semantics of OWL DL

In OWL Dictionary

$$V ::= V_L \cup V_{URI}$$

where V_L represents a set of literals and V_{URI} URI great number of references. The set of V_{URI} is created as follows:

- V_I, a set of separate names, for example, $Pavlo$;
- V_O, a set of names of ontologies, as a rule, consisting of the URL addresses specifying where ontologies are stored;
- V_{IC}, a set (personal-) class names, for example, $owl : Thing$, $owl : Nothing$;
- V_{DC}, a set of class names of data types, for example, $rdfs : Literal$, $xsd : gDay$, $xsd : integer$;
- V_{IP}, a set (personal-) names of properties, for example, has the father;
- V_{DP}, a set of names of property of data types, for example, height in meters;
- V_{AP}, a set of names of properties of summaries, for example, $owl : label$, $owl : seeAlso$;
- V_{OP}, a set of names of properties of ontologies, for example, $owl : import$.

Symbolic circuit in this case following: for use of a C class; for property P; for data type D; for a personal object I; for summaries of A; for ontology O. For users, familiar with FOL, the great number of V_I represents a set of separate constants; V_{IC} and V_{DC} represent sets of unary predicates (classes); V_{DP}, the V_{IP}, V_{AP} represent sets from dyadic predicates (properties). The single complexity of the OWL DL dictionary is that one - and dyadic predicates are subdivided depending on whether they belong to copies/objects, data types, summaries or ontologies. Besides, OWL the logical dictionary consists of characters for creation of classes, i.e., $\sqcap$, $\sqcup$, $\bar{}$, $\exists$, $\forall$, and also characters for a creation formula, i.e. $\sqsubseteq$ and .

Now we will pay attention to syntax of OWL, i.e. how to construct terms (atomic classes or difficult classes) and formulas (to OWL axioms and the facts).

The set of the classes OWL recursively is determined by the following rule:

$$QWL - Class ::= C|T| \perp |\bar{C}|C_1 \sqcap C_2|C_1 \sqcup C_2|\forall P.C|\exists P.C|\forall T.D|\exists T.D| \leq nPint| \leq nT|OneOf(i_1, ..., i_k)|OneOf(l_1, ..., l_k)$$

Where C designates a class (atomic or complex); $P \in V_I P$ atomic property, $D \in V_{DC}$ data type class; and $T \in V_{DP}$ property of data type. $Characters\>$; and $\perp$ aren't constrained by $owl : Thing$ and $owl : Nothing$. Pint means that property P isn't transitive property or isn't sub-property of transitive property. The predicate of $OneOf$ is a concept of the designer who provides lists of separate names i or literals of I.

Now we will provide OWL semantics. First, we will define the OWL model, namely function of interpretation for atomic classes, data types, properties, etc. Then we will continue this function of interpretation of difficult classes. And, at last, we will define truth conditions for OWL axioms and the relations of the logical investigation between ontologies.

OWL the M_{OWL} model represents a triple $\langle R, R_D, R_O, I_C, I_P, I_I, I_L \rangle$. The set of R is area of resources, with $R_O \subseteq R$ a set of objects or separate copies,

and $R_D \subseteq R$ a set of data types or literal values. Let's pay attention that $R_D \cup R_O = \oslash$. Every I_I is interpretation of function for classes, properties of separate names and literals. More precisely:

- $I_C(owl:Thing) = R_O \subseteq R$;
- $I_C(owl:Nothing) = \subseteq R$;
- $I_C(owl:Literal) = R_D \subseteq R$;
- $I_C : V_C \to \rho(R_O)$;
- $I_C : V_D \to \rho(R_D)$;
- $I_P : V_{DP} \to \rho(R_O \times R_D)$;
- $I_P : V_{IP} \to \rho(R_O \times R_O)$;
- $I_P : V_{AP} \cup rdf:type \to \rho(R \times R)$;
- $I_P : V_{OP} \cup rdf:type \to \rho(R \times R)$;
- $I_I : V_I \to R_O$;
- $I_L : V_L \to R_D$.

Below in tab.1 the recursive extension of difficult classes is given.

Table I. THE RECURSIVE EXTENSION OF DIFFICULT CLASSES

Syntax	Semantics
C	$R_O \setminus I_C(C)$
$C_1 \sqcap C_2$	$I_C(C_1) \cap I_C(C_2)$
$C_1 \sqcup C_2$	$I_C(C_1) \cup I_C(C_2)$
$\forall P.C$	$\lvert o \in R_O : \langle o, o' \rangle \in I_P(P) \Rightarrow o' \in I_C(C)$ for all $o' \rvert$
$\exists P.C$	$\lvert o \in R_O : \langle o, o' \rangle \in I_P(P) o' \in I_C(C)$, for some $o' \rvert$
$n \leq P$	$\lvert o \in R_O : \lvert o' : \langle o, o' \rangle \in I_P(P) \rvert \leq n \rvert$
$\forall T.D$	$\lvert o \in R_O : \langle o, d \rangle \in I_P(T) \Rightarrow d \in I_C(D)$ for all $d \rvert$
$\exists T.D$	$\lvert o \in R_O : \langle o, d \rangle \in I_P(T) d \in I_C(D)$, for some $d \rvert$
$n \leq T$	$\lvert o \in R_O : \lvert d : \langle o, d \rangle \in I_P(T) \rvert \leq n \rvert$
$OneOf(i_1, ...i_k)$	$\lvert (I_I(i_1), ..., I_I(i_k) \rvert$
$OneOf(l_1, ...l_k)$	$\lvert (I_L(l_1), ..., I_L(l_k) \rvert$

Consistencies and logical consequence fitting ontology are defined as follows: considering the dictionary V and ontology of $O ::= T - axioms \cup A - axioms$, we have

$M_{OWL} \rvert = O$ iff $M_{OWL} \rvert = \psi$, for all $\psi \in O$

and any linguistic element in O is supported in V

$O \rvert \neq \perp$ iff $M_{OWL} \rvert = O$, for some M_{OWL}

$O \rvert = O'$ iff $M_{OWL} \rvert = O \Rightarrow M_{OWL} \rvert = O'$, for all M_{OWL}

IV. SYNTAX AND SEMANTICS OF SWRL

A. OWL DL and ML Rules

In this section it will be a question about distribution of OWL DL on SWRL which represents a combination from OWL DL and ML Rules (Rule Markup Language). Rules are defined as: prior and posteriori. If all operators in the previous expression are defined as truthful, then all statements in a further expression applicable. Thus, new properties can be appropriated to copies, in the ontology based on a current status of the knowledge base. SWRL also defines library of embedded functions which can be applied to copies. They include numerical comparing, simple arithmetical actions and manipulation with lines, temporal functions.

Semantics is based by SWRL on OWL DL so doesn't support direct reasons about classes and properties. The rule SWRL contains the previous part which is mentioned as a body, and the following part which is mentioned as the head. Both the body and the head are formed from the positive conjunction of atoms:

$$atom \wedge atom... \Rightarrow atom \wedge atom$$

While SWRL doesn't support objections of atoms or an objection as a failure or a disjunction, it supports a classical objection. For example, programmer $(?P)$ represents atom where the programmer is the class name OWL, and $?P$ is replaceable that represents OWL an individual. Informally the rule SWRL can be read that if all atoms in a prior part truthful, then, the following part, also be truthful. There are seven types of atoms, a constant look $P(arg_1, arg_2...arg_h)$, that is predicate P_i his arguments:

- class atoms;
- atoms of properties of individuals;
- atoms of properties which are transferred on values, data;
- atoms of different individuals;
- atoms of similar individuals;
- built-in atoms;
- data span atoms.

DLP because it saves complete expressiveness of DL in addition with language of rules is suitable for obtaining bigger expressiveness, SWRL more, than. For this reason, strongly safe subset of SWRL as the best combination of OWL and rules as it saves solubility in case of minimization of losses in expressiveness is represented.

B. SHOIN (D)

Syntax of SHOIN (D) – DL is equivalent to syntax of OWL DL, and is finite, isn't sufficient to write rules. Thus, there is a need to rely on syntax of FOL. However, considering that SWRL is own extension of the OWL DL language, it doesn't make any sense to express SWRL of a formula partially in syntax of DL and partially in syntax of FOL. Fortunately, there is a simple transformation between the DL formulas and formulas of FOL [8]. Further we will assume that everything SWRL of a formula express in syntax of FOL, according to the equivalence given in [9,10].

The SWRL dictionary is OWL DL of the dictionary V with adding of sets:

- V_{IX} for the separate variables designated through x, y, z;
- V_{DX} for the variables of data type designated through m,n;
- $V_{built-in}$ for the built-in names.

Any variable in V_{IX} or a name in V_I will be called the term an object, and we will designate through t with indexes if

it is necessary. The term data type, we will designate through V, either a literal in V_L or a variable of data type in V_{DX}.

The logical SWRL dictionary expands OWL logical the dictionary with the help $\rightarrow$ and $\wedge$.

As SWRL is approved in FOL, its logical lexicon is equivalent to FOL, namely the functional sheaves $(\wedge, \vee, \rightarrow, \neg)$ and quantifiers $(\exists, \forall)$.

The set of SWRL atoms is determined by the following rule:

$$SWRL - atom ::= C(t)|D(v)|P(t_1, t_2)|T(t, v)|t_1 = t_2|t_1 \neq t_2$$

where With represents the class OWL (atomic or complex); $P \in V_{IP}$ OWL atomic property, $D \in V_{DC}$ class OWL data type; and $T \in V_{DP}$ property OWL data type.

The rule set of SWRL is the smallest set constructed of SWRL atoms, such where each element has the form:

$$A_1 \wedge ... \wedge A_2 \rightarrow A$$

where A_i and A SWRL atoms. And the head governed; and (perhaps empty) finite connection $A_1 \wedge ... \wedge A_k$ is a rule body. We will designate rules through r. There is k universal quantifiers which determine volume by all rule and it connects variables in the rule. X it is possible to set somehow as they are only universal quantifiers, and their changeover won't enter errors.

- classes OWL (i.e. their FOL conversion);
- OWL axioms and facts (i.e. their FOL conversion);
- SWRL of rules.

The SWRL model is designated through M_{SWRLg} expands the M_{OWL} models with function of assignment of

$$g ::= g_I \cup g_D$$

where:

- $g_I : V_{IX} \rightarrow R_O$;
- $g_D : V_{DX} \rightarrow R_D$.

The relations of SWRL expand OWL relations as follows:

$M_{SWRLg}| = C(t)$ iff $g_I \cup I_I(t) \in I_C(C)$

$M_{SWRLg}| = P(t_1, t_2)$ iff $g_I \cup I_I(t) \in I_C(C)$

$M_{SWRLg}| = t_1 = t_2$ iff $\langle g_I \cup I_I(t_1), g_I \cup I_I(t_2)\rangle \in I_P(=)$

$M_{SWRLg}| = t_1 \neq t_2$ iff $\langle g_I \cup I_I(t_1), g_I \cup I_I(t_2)\rangle \notin I_P(=)$

$M_{SWRLg}| = C(v)$ iff $g_D \cup I_L(v) \in I_C(C)$

$M_{SWRLg}| = T(t, v)$ iff $\langle g_I \cup I_I(t), g_D \cup I_L(v)\rangle \in I_P(D)$

$M_{SWRLg}| = A_1 \wedge ... \wedge A_2 \rightarrow A$

iff $M_{SWRLg}| = A_1 \wedge ... \wedge A_2 \Rightarrow M_{SWRLg}| = A$

C. *Protege-owl API*

Such popular development environment of ontologies as Protégé includes plag-in Swrltab, for creation and processing of the rules SWRL [3,10]. SWRL is supported by the moralist of Pellet to the place where rules can be defined as "Dl-safe".

Protégé [11] represents a flexible platform which prepares for development of arbitrary models of managed applications and components. It has an open architecture which allows programmers to integrate $plag_n$ which can appear in the form of separate inserts, specific components of the interface of the user ($v_d zheta$), or carry out any other tasks on the current model.

Protege provides several extension points where developers can dynamically add components as so-called plug-ins. The following fig.1 illustrates the types of plug-ins that you can create for the Protege-OWL editor.

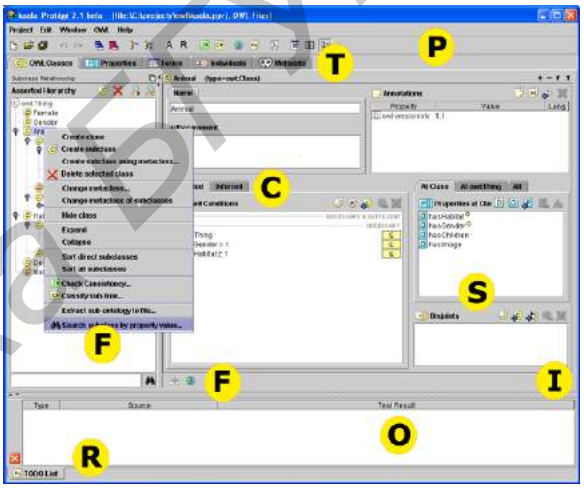

Figure 1. Plug-ins that can be created for the Protege-OWL editor

- S - Slot widget plug-ins are a Core Protege feature. A slot widget is a plug-in that can display and edit a property value on a form. Examples of default slot widgets include the list of disjoint classes, the conditions widget, and the annotation properties widget. You can create your own slot widgets and add them to the forms using the Forms tab.
- P - Project plug-ins are a Core Protege feature. They allow programmers to execute arbitrary code when a project is created, loaded, or closed. In particular, they can be used to add menus or toolbar buttons. They can also be used to attach arbitrary listeners to a knowledge base, such as agents.
- F - Resource action plug-ins can appear in the right-click menu of a selected class, property, or individual, or in the lower left corner of a form (as shown by the second 'F' above). A resource action plug-in must be a subclass of ResourceAction and you need to add an entry "ResourceAction=True" to your manifest file. Then, the ResourceAction is able to decide whether it wants to appear in the context menu, or also in the icon bar at the bottom left corner of a form.

- I - Resource display plug-ins can be used to add arbitrary components to the lower right corner of each form. You need to subclass ResourceDisplayPlugin and add an entry "ResourceDisplayPlugin=True" to your manifest file. Then you get a reference to a resource, e.g., an owl:Class, and a JPanel in which you can add buttons or other small components.
- O - Ontology test plug-ins are plug-ins that will be executed when the user presses the test ontology buttons. Each test must be a subclass of OWLTest and requires a manifest entry (check the Protege-OWL editor's manifest file for examples). The tests can return a test result object, which is then used to display results to the user.
- R - Result panel plug-ins are arbitrary components that can be displayed as a tab at the bottom of the screen. Examples include the "Find Usage" results, the classification output, and the ontology test results. You must subclass ResultsPanel and can then use some standard services such as selecting a highlighted object from there. You can add or remove your result panels as a result of some action using the ResultsPanelManager.
- C - Conditions widget extension plug-ins can be installed by a project plug-in to add additional tabs to the conditions widget. This is currently in its infancy, but you can call ConditionsWidget.addNestedWidgetFactory to add your panel, e.g., to display the abstract syntax.

The Protege Programming Development Kit (PDK) has a lot of general information on how to write, package, and distribute plug-ins. The best way to get started is to examine existing plug-ins that were written for the Protege-OWL editor, e.g., OWLViz, OWLDoc, Protege Wizards, etc. Pay particular attention to the manifest and protege.properties files for these plug-ins. The latter is needed for your plug-in to declare a dependency on the Protege-OWL editor.

Protege-owl API [12] is open source code of Java library for OWL and RDF(S). API provides classes and methods for loading and saving the OWL files, requests and handling the OWL models of data, and also for execution of reasons on the basis of the mechanism DL. Besides, API is optimized for implementation of the user graphic interfaces.

Jena [13] is one of the most widely used by the Java API, for RDF and OWL, providing services for representation of model, parse, persistence of the database, execution of requests and some instruments of visualization. Protege-owl API (v 3.4) and lower versions integrated with Jena, and the Jena ARP analyzer is used by Protégé-owl parcer.

The Jena inference subsystem is designed to allow a range of inference engines or reasoners to be plugged into Jena. Such engines are used to derive additional RDF assertions which are entailed from some base RDF together with any optional ontology information and the axioms and rules associated with the reasoner. The primary use of this mechanism is to support the use of languages such as RDFS and OWL which allow additional facts to be inferred from instance data and class descriptions. However, the machinery is designed to be quite general and, in particular, it includes a generic rule engine that can be used for many RDF processing or transformation tasks.

The overall structure of the inference machinery is illustrated at Fig.2.

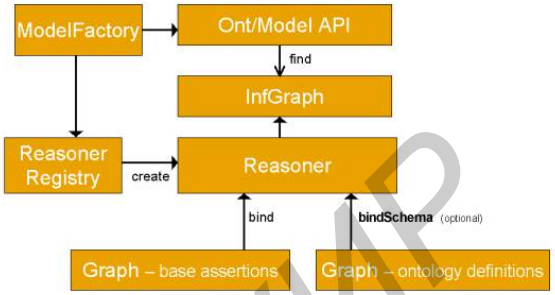

Figure 2. Overall structure of inference machinery

Applications normally access the inference machinery by using the ModelFactory to associate a data set with some reasoner to create a new Model. Queries to the created model will return not only those statements that were present in the original data but also additional statements than can be derived from the data using the rules or other inference mechanisms implemented by the reasoner.

As illustrated at the Fig. 2 the inference machinery is actually implemented at the level of the Graph SPI, so that any of the different Model interfaces can be constructed around an inference Graph. In particular, the Ontology API provides convenient ways to link appropriate reasoners into the OntModels that it constructs. As part of the general RDF API we also provide an InfModel, this is an extension to the normal Model interface that provides additional control and access to an underlying inference graph.

The reasoner API supports the notion of specializing a reasoner by binding it to a set of schema or ontology data using the bindSchema call. The specialized reasoner can then be attached to different sets of instance data using bind calls. In situations where the same schema information is to be used multiple times with different sets of instance data then this technique allows for some reuse of inferences across the different uses of the schema.

To keep the design as open ended as possible Jena also includes a ReasonerRegistry. This is a static class though which the set of reasoners currently available can be examined. It is possible to register new reasoner types and to dynamically search for reasoners of a given type. The ReasonerRegistry also provides convenient access to prebuilt instances of the main supplied reasoners.

Interfaces of the Protégé-owl model are located in an inheritance hierarchy. The review of available interfaces can be found in [14], with the basic interface of all RDF resources from which the received sub-interfaces for classes, properties and copies (objects).

There is an accurate discrepancy in model between the named classes and anonymous classes. The named classes are used for creation of separate copies while anonymous classes are used for determination of logical characteristics

(restrictions) from the called classes. Classes which are logically defined can be used for creation of difficult expressions from restrictions of a class and the named classes. As well as restrictions, logical classes make a sense only if they are connected to the defined named class or property.

In SWRL, predicate characters can include the classes OWL, properties or data types. Arguments can be separate copies or value of the data OWL, or replaceable, related. All replaceable in SWRL are considered as universal quantifiers, from them by restriction of volume of this rule.

The $built - in$ SWRL are predicates which recognize that they undertake this one or several evaluated arguments. A row of the basic embedded functions for mathematical and urgent operations contain in SWRL Proposal. These built-in modules are defined in the $swrlb.owl$ [15] file. By agreement, basic SWRL $swrlb$ space name qualifier can precede all.

V. Conclusion

The article discussed issues related of integration of OWL DL and rules were considered. The explanation of semantics of OWL DL with determination of the OWL model, namely – interpretation functions, and with determination of conditions of truth for OWL axioms and the relations is given. And in the inference the method of integration of OWL and rules with SWRL is provided.

Advantages of use of SWRL were given in article that at the moment is the most widely used language of rules in community Semantic Web. It is described possibilities of Protégé-owl AP I as the popular development environment of ontologies of Protégé includes plag-in Swrltab, for creation and processing of the rules SWRL. There was analyzed that SWRL is supported by moralists of Protégé to the place where rules can be defined as "Dl-safe".

References

[1] World Wide Web Consortium (W3C): OWL Web Ontology Language Guide [Electronic resource] // W3C Recommendation for a new version of OWL. – Mode of access: https://www.w3.org/TR/owl-guide. – Title from the screen.

[2] World Wide Web Consortium (W3C): OWL Web Ontology Language Guide [Electronic resource] // W3C technical report and recommendation for OWL. – Mode of access: https://www.w3.org/TR/owl-guide/. – Title from the screen.

[3] World Wide Web Consortium (W3C): OWL Web Ontology Language Guide [Electronic resource] // W3C technical report and recommendation for SWRL. – Mode of access: https://www.w3.org/TR/swrl-guide/. – Title from the screen.

[4] Rule Markup Language [Electronic resource] // Rule Markup Language. – Mode of access: http://www.ruleml.org. – Title from the screen.

[5] Breuker J., Use and reuse of legal ontologies in knowledge engineering and information management [Text] / J. Breuker, A. Valente, R. Winkels // Benjamins V. R. Law and the Semantic Web, LNAI 3369. – Berlin: Springer, 2005. – 36–64 pp.

[6] 6Motik B., Structured objects in OWL: Representation and reasoning [Electronic resource] / B. Motik, B. C. Grau, U. Sattler // WWW 2008, Refereed Track: Semantic - Data Web - Semantic Web, April 21-25, 2008: Beijing, China. – Mode of access: WWW.URL: https://www.cs.ox.ac.uk/files/4557/p555-motikA.pdf. - Last access: 2012. – Title from the screen.

[7] Khala K. The description of a method of integration of OWL DL and rules with sequential presentation [Text] / K. Khala // Intelligent analysis of information, research papers of XVI International Conference, IAI-2016 behalf T.A.Taran (May 18-20, 2016., Kyiv).-K., 2016. - 274-280 pp.

[8] Volz R., Web Ontology Reasoning with Logic Databases [Text]: PhD thesis Institute AIFB, University of Karlsruhe, 17.02.04 / Prof. Dr. Rudi Studer. – Karlsruhe, 2004. – 287p.

[9] World Wide Web Consortium (W3C): Semantic Web Rule Language Combining OWL and RuleML [Electronic resource] // W3C proposal for a SWRL. – Mode of access: https://www.w3.org/Submission/2004/SUBM-SWRL-20040521. – Title from the screen.

[10] SWRLJessTab Protégé plug-in [Electronic resource] // Plug-in for Protégé Editor. – Mode of access: http://protege.cim3.net/cgi-bin/wiki.pl?SWRLJessTab. - Last access: 2015. – Title from the screen.

[11] Protégé [Electronic resource] // A free, open-source ontology editor and framework for building intelligent systems. – Mode of access: http://protege.stanford.edu/. – Title from the screen.

[12] 12Protégé-OWL API [Electronic resource] // Protégé-OWL API. – Mode of access: http://protege.stanford.edu/plugins/owl/api/. – Title from the screen.

[13] Apache Jena [Electronic resource] // Apache Jena. – Mode of access: http://jena.sourceforge.net/. – Title from the screen.

[14] Sanchez-Macián, A., Pastor, E., Vergara, J. de L., Lopez, D. (2007). Extending SWRL to Enhance Mathematical Support. Web Reasoning and Rule Systems (p. 358–360). Retrieved from http://dx.doi.org/10.1007/978-3-540-72982-2-30.

[15] SWRL swrlb [Electronic resource] // SWRL swrlb. – Mode of access: http://www.w3.org/2003/11/swrlb. – Title from the screen

ПОСЛЕДОВАТЕЛЬНОЕ ИЗЛОЖЕНИЕ МЕТОДА ДЛЯ ИНТЕГРАЦИИ OWL DL И SWRL С ИСПОЛЬЗОВАНИЕМ PROTÉGÉ-OWL API

Хала Е.А.

Эта статья посвящена описанию метода интеграции OWL DL и правил, с последовательным изложением. В статье описывается необходимость подобной интеграции, и анализируются семантики OWL DL, а также приводиться метод по интеграции OWL и правил с использованием специального языка правил SWRL.