EasyChair Preprint

№ 10517

Comparison of Ontology Reasoning Systems for Semantic Web Expert System

Olegs Verhodubs

July 8, 2023

Comparison of Ontology Reasoning Systems for Semantic Web Expert System

Olegs Verhodubs

***Abstract* – The purpose of this paper is to analyze and to compare different ontology reasoning systems. Reasoning system or inference engine is one of main components of any expert system. This comparison of reasoning systems is necessary for SWES (Semantic Web Expert System) construction [1]. SWES is an expert system, which will be able to process ontologies from the Web, to supplement or to develop its knowledge base [1]. Available publications describe the problem of ontology reasoning systems comparison for specific purposes that is why it is necessary to make such a comparison for SWES.**

Keywords – Semantic Web, Expert Systems

I. Introduction

There were developed a lot of expert systems. All of them were developed for different purposes and had its particular qualities. However all expert systems have a typical structure. Such an expert system structure consists of the following elements [1]:

- Working memory;
- Knowledge base;
- Inference engine;
- Knowledge acquisition component;
- Explanation component;
- Dialogue component.

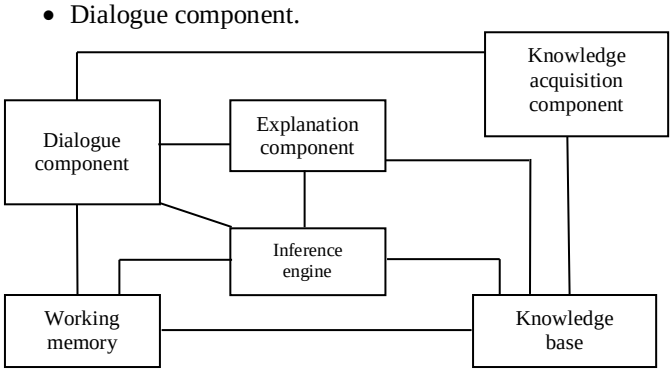

Fig. 1. The typical structure of a static expert system.

Each element of an expert system has its own task. Working memory is necessary for data storing which is used for a current task solving. Knowledge base is necessary for knowledge storage which describes a domain. Inference engine is a program that models expert's style of reasoning using knowledge from the knowledge base. Knowledge acquisition component automates the process of an expert system filling with data which is executed by an expert. Explanation component explains how the system executed the task solution and what knowledge was used that could facilitate testing and increase trust in the results. And dialogue component is focused on interaction with users to give the possibility of knowledge input and to show the results of task solution.

Inference engine is one of the major components of any expert system, which utilizes knowledge base rules for generation of new facts. There are a lot of inference engines for the Semantic Web, and they are usually called as semantic reasoners. Among the most famous semantic reasoners are Bossam, Jena, Pellet, KAON2, Fact and others [2]. All of them have different features: some of semantic reasoners are free, while others are non-free; some of semantic reasoners utilize one reasoning algorithm, while others use another reasoning algorithm; some of semantic reasoners are able to process OWL (Web Ontology Language) constructs [3], while other semantic reasoners are able to process RDF (Resource Description Framework) [4] triples only. All of inference engines are useful, but one of them is better for one task and another inference engine is better for other tasks. The main purpose of this paper is to compare available inference engines for choosing the most appropriate one to implement in SWES. SWES is an expert system, which will be capable to use OWL ontologies from the Web, to extract rules from them and to supplement its knowledge base in automatic mode. SWES developing is the final goal of the research.

The paper is organized as follows. Section II gives an overview of related works. Section III lists several semantic reasoners and gives an overview of them. Section V analyzes SWES requirements for semantic reasoners. And finally conclusion is presented.

II. Related works

There are a lot of papers, which describe ontology reasoning systems. Some papers present well-defined ontology reasoning systems while the other present several ontology systems and compare their capabilities. There are papers, where are described some being developed applied systems, which exploit certain reasoning systems. All these papers can serve as a useful material for assistance in choosing ontology reasoning system for SWES. Let us overview several papers.

In [5] paper is described DR-Prolog system. This system is based on Prolog and is designed to answer queries. The system can reason with rules and ontological knowledge. DR-Prolog is compatible with RuleML (Rule Markup Language) [6].

In [7] are presented several rule engines as Jess (Java Expert System Shell), XSB (Logic Programming and Deductive Database System, which extends Prolog), Cvm (general-purpose data processor for the Semantic Web), Mandarax (the first complete input-processing-output environment for RuleML) and three rules-enabled ontology engineering platforms as KAON (an open-source ontology management infrastructure including a comprehensive tool suite to ease the creation and management of ontologies and to assist the development of ontology-based applications), FLORA-2 (a sophisticated object oriented knowledge base language and application development environment) and the Instance Store (a Java application for performing efficient and scalable DL (Description Logic) reasoning over individuals). Here are also briefly described several OWL reasoners. The main goal of the paper is to propose OWL2Jess that is a hybrid reasoning framework for converting OWL ontologies to Jess knowledge base.

Paper [8] presents different semantic reasoners as FACT++ (DL reasoned), RacerPro (implementation of tableau calculus), Pellet (reasoner based on well-known tableau algorithms for T- and ABox reasoning), KAON2 and others. Before the existing semantic reasoners evaluations are listed the following evaluation criteria:

- Language conformity;
- Correctness;
- Efficiency;
- Interface capabilities;
- Inference services.

Language conformity means conformity to the existing official OWL specification that is checking which OWL constructs a system is able to handle. Correctness is soundness and completeness of the systems. Efficiency means runtime and resource consumption of a few realistic ontologies as well as artificially compiled samples with different complexity. Interface capabilities are interactive communication vs. batch-processing, support for loading URLs, programming interface, client-server architecture etc. Inference services offered system services and system handling.

Listed semantic reasoners are analyzed according to mentioned evaluation criteria in details. These semantic reasoners are also tested using thought-out system of varied tests.

Paper [5] describes DLEJena, which is a practical reasoner. DLEJena combines the forward-chaining rule engine of Jena and the Pellet DL reasoner. This combination is based on rule templates, instantiating at run-time a set of ABox OWL 2 RL/RDF Jena rules dedicated to a particular TBox that is handled by Pellet. The goal of DLEJena is to handle efficiently, through instantiated rules, the OWL 2 RL ontologies under direct semantics, where classes and properties cannot be at the same time individuals. It is stated that DLEJena achieves more scalable ABox reasoning than the direct implementation of the OWL 2 RL/RDF rule set in the Jena's production rule engine, which is the main target of the system.

Paper [9] gives an overview of Jena, which is a Java framework for building Semantic Web applications and other semantic reasoners as Sesame (an open source RDF framework with support for RDF Schema inferencing and querying), Mulgara (an open source, scalable, transaction-safe, purpose-built database for the storage and retrieval metadata), Redland (a set of C libraries that provide support for RDF). This paper is useful in the sense that it describes architectures, storage and querying systems of mentioned reasoners.

Paper [10] is completely devoted to Pellet. It defines Pellet as the first sound and complete OWL-DL reasoner with extensive support for reasoning with individuals, including nominal support and conjunctive query, user-defined datatypes and debugging support for ontologies. It is indicated that Pellet is DL reasoner, based on tableaux algorithms. The tableaux algorithm checks the consistency of a knowledge base and all the other reasoning services are reduced to consistency checking.

Presentation [11] gives comparison of ontology reasoning systems using custom rules. Here there are examined Jena, Pellet, KAON2, Oracle 11g, OWLIM. There are used nine datasets with different size. Setup time (this stage includes loading and preprocessing time before any query can be made) and query processing time (this stage starts with parsing and executing the query and ends when all the results have been saved in the result set) are metrics, exploited for testing ontology reasoning systems. Plenty of diagrams in the presentation allow understanding the results of Jena, Pellet, KAON2, Oracle 11g and OWLIM testing.

Paper [12] presents all-in-one survey of the Semantic Web in terms of ontologies. Starting with the term "ontology", there are made demands for any ontology to realize it explicit and useful for reuse. Then evolution of the Semantic Web ontology languages is shown from RDF to OWL. Further ontology editors are presented namely Protégé and SWOOP. Hereupon ontology repositories are listed. This paper also describes OWLJessKB, Java Theorem Prover (JTP), Jena, F-OWL, FACT++, Racer, Pellet, TRIPLE, SweetRules, which are nominated as ontology language processors. After that three kinds of inference are reflected (forward chaining, backward chaining and hybrid chaining). Then ontology-based information integration is discussed. And the last part of the paper is dedicated to the Semantic Web ontology usage.

There are a lot of other papers, which describe mentioned and not mentioned ontology reasoning systems. But it makes no sense to mention all of them, because mostly they repeat each other. So, it is necessary to collect detailed information about major ontology reasoning systems to choose the best one for combining in SWES.

III. SEMANTIC REASONERS

In previous section there were described plenty of ontology reasoning systems. But regardless of their

characteristics all of these ontology reasoning systems can be divided into two main categories:

- Multi-purpose ontology reasoning systems, and
- Highly specialized ontology reasoning systems.

Multi-purpose ontology reasoning systems are the systems, which are designed for general use in different applications and having a set of universal properties. On the contrary, highly specialized ontology reasoning systems are the systems, which are designed for implementation at specific projects, only. It is clear that highly specialized ontology reasoning systems have a limited set of properties, selected for the needs of one or two projects. Obviously, it is necessary to choose among multi-purpose ontology reasoning systems for implementing in SWES, because SWES needs in powerful reasoner, whose properties may be needed not only in being developed edition of the project, but also in possible future editions of this project. Highly specialized ontology reasoning systems generally have features for current edition of the system, only. That is why in this section only multi-purpose ontology reasoning systems are discussed.

There are plenty of multi-purpose ontology reasoning systems, but here will be considered only seven of them, because they are most often cited in other research papers:

- Bossam;
- FaCT++;
- Hermit;
- Jena;
- Kaon2;
- Pellet;
- RacerPro;

Bossam is a forward-chaining inference engine for the Semantic Web. It is basically a RETE-based rule engine with native supports for reasoning over OWL ontologies, SWRL ontologies, and RuleML rules. Bossam does not support SPARQL and has the following expressivity features:

- URI references as symbols;
- 2-nd order logic syntax;
- disjunctions in the antecedent and conjunctions in the consequent;
- URI-based java method attachment;
- Support for both negation-as-failure and classical negation.

Bossam test version is available for download, and it works with Java programming language.

FaCT++ (Fast Classification of Terminologies) is an OWL DL reasoner with ABox and nominal reasoning support. It is also a tableaux-based reasoner with backward chaining. This reasoner started as C++ re-implementation of FaCT reasoner at the University of Manchester and could be used with C++ programming language. FaCT++ covers OWL as well as OWL2 excluding support for key constraints and some data types. FaCT++ provides standard TBox reasoning tasks like subsumption and consistency checking as well as taxonomy construction. It also performs instance classification. Unfortunately, FaCT++ does not support any kind of query languages. It is used as one of the default reasoners in the Protégé OWL editor. FaCT++ is available for download both as a binary file and as source code. To build FaCT++ you will need C++ compiler.

HermiT is the first publicly-available OWL reasoner based on a novel “hypertableau” calculus, which provides much more efficient reasoning than any previously-known algorithm. HermiT is the first reasoner able to classify a number of ontologies which had previously proven too complex for any available system to handle. HermiT can determine whether or not the ontology is consistent, identify subsumption relationships between classes, and much more. The reasoner supports DL Safe rules without SWRL built-in atoms. Additionally, HermiT supports reasoning with description graphs and description graph rules. HermiT is available as an open-source Java library and includes both a Java API and a simple command-line interface.

Jena is a Java framework for constructing Semantic Web applications. The Jena framework comes with a complete set of reasoners:

- An RDFS reasoner;
- An OWL reasoner;
- A transitive reasoner;
- A generic rule-based reasoner.

There are three variants of the RDFS reasoner. The Simple reasoner only implements the transitive closure of the subClassOf and subPropertyOf relations, the entailments regarding the range and domain of properties and the implication of subPropertyOf and subClassOf. The Default reasoner additionally includes the axiomatic rules and the Full reasoner implements almost all the RDFS axioms and closure rules.

The OWL reasoner of the Jena framework supports only OWL-Lite language. For OWL DL reasoning can be used an external DL reasoner such as Pellet, Racer or FaCT. The Jena OWL reasoner applies rules to propagate OWL implications over instance data (i.e. the ABox of a DL knowledge box). Class reasoning, i.e. reasoning over the TBox of a DL knowledge base, is implemented similarly through the generation of an instance. Inferences are computed at the instance level, and class reasoning is deduced from these inferences. There are three implementations with different support for OWL entailments that differently impact on the cost and efficiency of reasoning. These implementations are full, mini and micro.

The Transitive reasoner of the Jena framework provides support for storing and traversing class and property lattices. This implements just the transitive and symmetric properties of rdfs:subPropertyOf and rdfs:subClassOf. It is not all that exciting on its own but is one of the building blocks used for the more complex reasoners. This engine is useful to perform a high-performance transitive closure over class and property hierarchies, and much more efficient than using the rule-based engines (Builtin RDFS or OWL reasoners).

Jena has a general purpose rule-based reasoner, which is used to implement both the RDFS and OWL reasoners but is also available for general use. This reasoner provides forward chaining, backward chaining and a hybrid execution model. In general, there are two internal rule engines one forward chaining RETE engine and one tabled datalog engine - they can be run separately or the forward engine can be used to prime the backward engine which in turn will be used to answer queries.

Jena also has its own storage subsystem that, if necessary, has to be installed separately. Jena storage subsystem has two species: TDB and SDB. TDB is a lightweight, scalable non-transactional storage and SPARQL query layer. TDB can be used as a high performance RDF store on a single machine. A TDB store can be accessed and managed with the provided command line scripts and via the Jena API. SDB is a SPARQL database subsystem for Jena. It provides for large scale storage and query of RDF data sets using conventional SQL databases. An SDB store can be accessed and managed with the provided command line scripts and via the Jena API.

Jena was an open-source project and was developed at HP Labs.

KAON2 is an infrastructure for managing OWL-DL, SWRL, and F-Logic ontologies. It was produced by the joint effort of the following institutions:

- Information Process Engineering (IPE) at the Research Center for Information Technologies (FZI);
- Institute of Applied Informatics and Formal Description Methods (AIFB) at the University of Karlsruhe;
- Information Management Group (IMG) at the University of Manchester.

KAON2 has the following features:

- An API for programmatic management of OWL-DL, SWRL, and F-Logic ontologies;
- A stand-alone server providing access to ontologies in a distributed manner using RMI;
- An inference engine for answering conjunctive queries (expressed using SPARQL syntax);
- A DIG interface, allowing access from tools such as Protégé;
- A module for extracting ontology instances from relational databases.

KAON2 is known to perform very well on large ABoxes, t.i. it aims at reasoning with large amounts of individuals [8]. It is short on elaborated TBox services [8].

KAON2 is available as a precompiled binary Java distribution and is free of charge for research and academic purposes. A commercial version has to be licensed from Ontoprise GmbH [8].

Pellet is an OWL-DL reasoner based on the tableaux algorithms developed for expressive Description Logics [8]. Pellet covers all of OWL-DL including inverse and transitive properties, cardi-nality restrictions, datatype reasoning for an extensive set of built-ins as well as user defined simple XML schema datatypes, enumerated classes (a.k.a, nominals) and instance assertions [10]. It is the first sound and complete OWL-DL reasoner with extensive support for reasoning with individuals including nominal support and conjunctive query, user-defined datatypes, and debugging support for ontologies. It implements several extensions to OWL-DL including combination formalism for OWL-DL ontologies, a non-monotonic operator, and preliminary support for OWL/Rule hybrid reasoning [10]. Pellet has the following inference services [8]:

- detection of unsatisfiable classes;
- checking for entailed statements;
- building the class taxonomy;
- ABox realization by showing classified individuals in the class hierarchy;

An exceptional property of Pellet is its ontology analysis and repair feature trying to convert OWL-Full ontologies into OWL-DL [8]. It also bears some non-standard debugging features as well as ontology partitioning functionality based on the e-connection calculus. Pellet also supports the conjunctive query languages SPARQL and RDQL [8].

Pellet is written in Java [10] by Evren Sirin and Bijan Parsia from the University of Maryland and is open source [8]. It provides various interfaces including a commandline interface, DIG server implementation, and reasoner API for Jena and the OWL API [8].

RacerPro is based on a tableau calculus and supports multiple T- and ABoxes [8]. RacerPro incorporates all optimization techniques of FaCT as well as some others for dealing with number restrictions and ABoxes [8]. RacerPro can reason about OWL-Lite knowledge bases, as well as OWL-DL with approximations for nominals, together with some algebraic reasoning beyond the scope of OWL [8]. Nominals in class definitions are approximated in a way which provides sound but incomplete reasoning. This reasoner also allows switching the UNA (unique name assumption) on or off [8]. RacerPro is able to reason with datatypes of type String, Integer, and Real. Similar to FaCT++ it currently does not support the newly introduced role expressions of OWL 1.1 [8]. Prototypical interface implementations for Java, CommonLisp and C++ are available [8].

RacerPro is developed by Volker Haarslev from Concordia University in Canada and by Ralf Moller from Hamburg University of Science and Technology Software, Technology, and Systems in Germany [8]. In general, RacerPro offers broad and flexible interfaces and inference services. However, there are some minor implementation flaws which result in problems or unexpected outcomes when dealing with multiple TBoxes or retracting given TBox statements [8]. In addition, this reasoner is non-free and is available for preview only.

IV. SEMANTIC REASONER FOR SWES

It is necessary to compare semantic reasoners listed in the previous section to reasonably choose the most appropriate one for the implementation of SWES. According to this purpose there are worked out several criteria for making this comparison. It should be noted that the criteria are aimed to semantic reasoners evaluation in terms of practical implementation in SWES instead of semantic reasoners technical features. Primarily this is due to the purpose of the research to realize Semantic Web Expert System or Semantic Web Expert System prototype. So, these criteria are the following:

- Licensing;
- Organization;
- Documentation;
- Programming language;
- Consistency checking;
- Reasoning features;
- Rule support;
- Storage subsystem.

Let us explain mentioned semantic reasoners evaluation criteria. The first criterion is licensing, and it shows if semantic reasoner is free or non-free software. Obviously free software is much more preferable, especially in the area of Semantic Web technologies, because this area is rather new and on the one hand free software may gain a stronger position in the community of the Semantic Web developers and thus become standard, but on the other hand strong position in the community of developers or software users increases motivation to develop this software in the future, what we are very interested in. The second criterion namely organization means organization, which elaborates semantic reasoner. The more respectable organization is the better for SWES. It is so, because authority of the organization apparently increases the chances of semantic reasoner support in the future. It seems very important because Semantic Web technologies will continue to evolve, and this will require the support of the semantic reasoner. The next criterion is documentation or how well semantic reasoner is documented. Qualitative documentation may replace many hours of persistent work on studying the properties of semantic reasoner. And in general this parameter is rather informative because it not only describes semantic reasoner features, but also product quality at all stages of its development. The fourth semantic reasoner criterion is programming language that is programming language semantic reasoner works with. Despite the abundance of programming languages, C++ and Java are basic programming languages among semantic reasoners. Of course they have their own advantages and disadvantages, however Java is preferred compared to C++. So, Java programming language is chosen because it is quite easy to use, Java is aimed at net software development, there are a lot of useful libraries created for Java and also developers, who know C++ may easy shift to Java, but not vice versa. If the above described criteria referred to the external characteristics of semantic reasoners, then the following criteria will refer to their functional features. The first such a criterion is consistency checking. As it is known each semantic reasoner works with OWL ontology. But ontology as well as any other data may have different quality. Some ontologies may have the correct syntax, others not. In addition some of the ontologies may have inconsistencies, and others may not have them. Consistency checking is an operation which can determine whether the ontology to be defective or not. So, this operation is very important, because it always precedes reasoning, and it can answer the question of whether the ontology to be processed semantic reasoner or not. The next criterion to distinguish different semantic reasoners is actually their reasoning features. These reasoning features may vary from one semantic reasoner to another semantic reasoner. The fact that the Semantic Web technologies are new technologies and it means that here changes occur very quickly. It is natural that some developers of semantic reasoners have time to quickly put these changes into their products, but not others. For example, OWL ontology format did not develop uniformly in time. First, XML format or specification appeared. Then there was RDF specification. After that RDFS specification came. And then OWL specification was worked out. In fact, the same OWL specification is not homogenous. This OWL specification includes three variants of OWL with different levels of expressiveness. Here are OWL variants [3]:

- OWL Lite;
- OWL DL;
- OWL Full.

At that OWL DL is more expressive than OWL Lite and OWL Full is more expressive than OWL DL. So that is common situation when different semantic reasoners support different specifications. That is one semantic reasoners can work i.e. reason with RDFS ontologies only, other semantic reasoners can work with OWL Lite and partially with OWL DL. But clearly the more specifications are supported with semantic reasoner, the better. Perhaps this problem will disappear in the future when the Semantic Web technologies when they reach the higher levels of development. One more criterion is rule support and this is one of the most important criteria to character semantic reasoners. There are several rule formats and the most known of them are:

- RuleML (Rule Markup Language);
- SWRL (Semantic Web Rule Language);
- R2ML (REWERSE Rule Markup Language).

Moreover there are plenty of own rule formats which exist only within the frameworks of semantic reasoners. In general it is necessary to mention that rule support in semantic reasoner is vital for SWES because its main idea is to process ontologies from the Web, to extract rules and to supplement SWES knowledge base with extracted rules in automatic mode. That is why semantic reasoner cannot be chosen for the implementation of SWES if it does not

support rules. The last semantic reasoner criterion is storage subsystem that shows if semantic reasoner has its own storage subsystem. Storage subsystem of semantic reasoner seems very useful because it allows preserving semantic data in convenient way specifically for semantic reasoner work.

Now that all of the criteria are listed and explained in details, let us characterize semantic reasoners according to these criteria in the pivot table to select one semantic reasoner for implementation in SWES:

TABLE I
Comparison of Semantic Reasoners

Semantic reasoner	Bossam	FaCT++	Hermit	Jena	KAON2	Pellet	RacerPro
Licensing	Non-free	Free	Free	Free	Free	Free	Non-free
Organization	Minsu Jang	University of Manchester	Oxford University	HP Labs	Universities of Manchester and Karlsruhe	Clark & Parsia, LLC	Racer Systems GmbH & Co. KG
Documentation	poorly-documented	well-documented	well-documented	well-documented	poorly-documented	well-documented	well-documented
Programming language	Java	C++	Java	Java	Java	Java	Java, Lisp
Consistency checking	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Reasoning features	OWL	OWL	OWL	RDFS, OWL, rule-based	OWL, F-logic, rule-based	OWL	OWL
Rule support	Yes	No	Yes	Yes	Yes	Yes	Yes
Storage subsystem	No	No	No	Yes	No	No	No

As can be seen from table I Jena is the best semantic reasoner according to the mentioned criteria because it is free, well-documented, uses Java programming language, has consistency checking function, has rule support and has its own storage subsystem. But the main advantage is that Jena has several reasoners what is very useful because it permits to use its own reasoner for specific data. One more unnamed Jena advantage is ability to change its working modes of rule-based reasoner. There are three such modes: forward chaining engine, backward chaining engine and hybrid rule engine. And these modes allow adjusting the order of rule processing. Extremely important ability for SWES is the possibility of combining OWL and custom rules inference. This is a key point in the project of SWES, because it enables to use different sources of knowledge extraction to construct really useful expert system for users. Of course Jena has several disadvantages, but first, they are not essential and second, they can be corrected in the future. One of Jena disadvantage is that it does not support SWRL specification, which combines OWL DL and OWL Lite sublanguages of OWL and also RuleML. It is necessary to mention that now Jena rule format fully satisfies the requirements of SWES, but it is possible that SWRL will be widely used in the Web in the future and then Jena facilities should be developed. On the other hand Pellet can be used in conjunction with Jena, and it supports SWRL specification. One more Jena problem is that Jena OWL reasoner is incomplete for OWL DL, however this problem can be easily solved, too. For this purpose developers of Jena advice using an external DL reasoner such as Pellet, Racer or FaCT.

It is clear that the theoretical analysis and comparison of semantic reasoners capabilities is necessary but not sufficient. Ideally you should test each semantic reasoner in practice. However it is not always possible because of many objective reasons. In one case, the reason is non-free software. In other cases, these reasons are lack of documentation or time limit to be able to test each semantic reasoner in details. Therefore there were tested several semantic reasoners only. Among them are Jena and Pellet. And in terms of practical use, Jena has established itself very well, because it has clear system of settings and functions.

V.CONCLUSION

So, as a result of the work has been selected the best semantic reasoner to be implemented in SWES namely Jena. Some actions have been taken to achieve this goal. First, there was analyzed future SWES work from the point of view of the user to detect the requirements for SWES semantic reasoner. Second, there were detected all available sources of data dedicated to the semantic reasoners. Third, the most intelligent sources of data were explored, and possible candidates for SWES semantic reasoner were identified. Then, in accordance with the requirements, identified during the first step, set of criteria was worked out to select the most appropriate semantic reasoner for SWES. And finally, all detected candidates for SWES semantic reasoner were analyzed for compliance with listed criteria and the most appropriate semantic reasoner was found. After the choice was made in favor of Jena, there were parsed additional advantages and disadvantages of Jena. Considering several Jena disadvantages, some remedies to neutralize these defects were proposed.

In general this paper continues sequence studies in the SWES project. In previous studies the conception of

Semantic Web expert system was founded and developed. The idea of rules extraction from OWL ontologies was presented and described in details, too. Then, this idea was realized practically, that is, the algorithm of OWL ontology transformation to rules [13] was implemented using Java programming language. And this paper demonstrates substantiation for the selection of semantic reasoner for SWES. It is easily seen that there are produced plenty of investigations; however it is necessary to take note that a lot of other studies have to be done for SWES implementation. One such study is to create ontology search algorithm in the Web according to the user's query. Here the main subtask is in the search of ontologies in the Web, but the other subtask is identification of the subject area from the user's query and also mapping ontologies for similarities. One more future study to be done is to consider and to describe the whole process of the search of solution from time of receiving of user's task to the time of output results. Here are a lot of pitfalls and this task has to be seriously investigated. Of course, there exist other minor tasks, but they will be discussed in the next papers.

REFERENCES

[1] O. Verhodubs, J. Grundspeņķis, Towards the Semantic Web Expert System. Riga: RTU Press, 2011. [Online]. https://ortus.rtu.lv/science/en/publications/12370/fulltext. [Accessed: September 17, 2012]

[2] "Category: Reasoner – Semantic Web Standarts", 2009. [Online]. Available: http://www.w3.org/2001/sw/wiki/Category:Reasoner. [Accessed: September 17, 2012].

[3] "OWL Web Ontology Language Guide", 2004. [Online]. Available: http://www.w3.org/TR/owl-guide/. [Accessed: September 17, 2012]

[4] "RDF – Semantic Web Standarts", 2009. [Online]. Available: http://www.w3.org/RDF/. [Accessed: September 17, 2012]

[5] G. Meditskos, N. Bassiliades, DLEJena: A Practical Forward-Chaining OWL2 RL Reasoner Combining Jena and Pellet. Elsevier Science Publishers B. V. Amsterdam, The Netherlands, 2010. [Online]. http://lpis.csd.auth.gr/publications/meditskos-jws09.pdf. [Accessed: September 17, 2012]

[6] "The Rule Markup Initiative". [Online]. Available: http://ruleml.org/. [Accessed: September 17, 2012]

[7] J. Mei, E. P. Bontas, Reasoning Paradigms for OWL Ontologies. Freie University of Berlin, Berlin, Germany, 2004. [Online]. http://edocs.fu-berlin.de/docs/servlets/MCRFileNodeServlet/FUDOCS_derivate_000000000422/2004_12.pdf;jsessionid=12AB068160C9504CACEB94149E22EE96?hosts=. [Accessed: September 17, 2012]

[8] T. Liebig, Reasoning with OWL. Ulm University, Ulm, Germany, 2006. [Online]. http://www.informatik.uni-ulm.de/ki/Liebig/papers/TR-U-Ulm-2006-04.pdf. [Accessed: September 17, 2012]

[9] J. Dokulil, J. Yaghob and F. Zavoral, Semantic Infrastructures. Intech. [Online]. http://cdn.intechopen.com/pdfs/9390/InTech-Semantic_infrastructures.pdf. [Accessed: September 17, 2012]

[10] E. Sirin, B. Parsia and others, Pellet: A Practical OWL-DL Reasoner. Elsevier Science Publishers B. V. Amsterdam, The Netherlands, 2007. [Online]. http://pellet.owldl.com/papers/sirin05pellet.pdf. [Accessed: September 17, 2012]

[11] H. Shi, K. Maly, S. Zeil, M. Zubair, Comparison of Ontology Reasoning Systems Using Custom Rules, 2011. [Online]. http://wims.vestforsk.no. [Accessed: September 17, 2012]

[12] L. Ding, P. Kolari, Z. Ding, S. Avancha and others, Using Ontologies in the Semantic Web: A Survey. Integrated Series in Information Systems, 2005. [Online]. http://ebiquity.umbc.edu/_file_directory_/papers/209.pdf. [Accessed: September 17, 2012]

[13] O. Verhodubs, J. Grundspeņķis, Evolution of Ontology Potential for Generation of Rules. Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics, Craiova, Romania, 2012.

Олег Верходуб. Сравнение онтологических машин вывода для реализации в Экспертной Системе Семантической Сети

Данная статья является очередным исследованием, которое связано с разработкой Экспертной Системой Семантической Сети (SWES). Если в предыдущих статьях был описан прототип внутренней структуры Экспертной Системы Семантической Сети, а также механизм генерации правил из конструкций OWL онтологий, то лейтмотив этой статьи – это рассмотреть существующие онтологические машины вывода и выбрать наиболее подходящую, чтобы включить и использовать ее в конструкции Экспертной Системы Семантической Сети. Для достижения этой цели были выполнены следующие действия. Для начала были исследованы в Интернете доступные статьи, которые посвящены этой тематике. В результате исследования соответствующих статей были выявлены несколько онтологических систем вывода как Bossam, FaCT++, Hermit, Jena, KAON2, Pellet, RacerPro, OWLAPI, Sesame и другие. Затем, после исследования этих статей и выявления нескольких онтологических машин вывода, были разработаны критерии для онтологической машины вывода, предназначенной для реализации в Экспертной Системе Семантической Сети. Вот эти критерии: вид лицензии, разработчик, качество документации, используемый язык программирования, наличие валидации онтологии, характеристика функций вывода, поддержка правил и наличие подсистемы хранения. Далее были выбраны семь наиболее известных многофункциональных онтологических машин вывода (Bossam, FaCT++, Hermit, Jena, KAON2, Pellet, RacerPro), которые были проанализированы в соответствии с выдвинутыми критериями. В результате этого анализа было решено, что Jena является наиболее подходящей онтологической машиной вывода для реализации в Экспертной Системе Семантической Сети. Конечно, не было возможно выбрать онтологическую машину вывода без ее практического опробования. В связи с этим практически были опробованы Jena и Pellet. С практической точки зрения Jena зарекомендовала себя с наилучшей стороны.