VERSITA
Applied Computer Systems
doi: 10.2478/acss-2013-0010
2013/ 14

Algorithm of Ontology Transformation to Concept Map for Usage in Semantic Web Expert System

Olegs Verhodubs, *Riga Technical University*, Janis Grundspenkis, *Riga Technical University*

***Abstract* – The main purpose of this paper is to present an algorithm of OWL (Web Ontology Language) ontology transformation to concept map for subsequent generation of rules and also to evaluate the efficiency of this algorithm. These generated rules are necessary to supplement and even to develop SWES (Semantic Web Expert System) knowledge base. This paper is a continuation of the earlier research of OWL ontology transformation to rules.**

Keywords – Expert Systems, Semantic Web.

I. Introduction

Nowadays there are a lot of ontologies in the Web and most of them are created by using OWL. It is obvious that ontologies become a resource that has to be utilized. Ontologies consist of concepts (classes), relations (or properties), instances and axioms and from the mathematical point of view, each ontology can be expressed as a 4-tuple < C, R, I, A >, where C is a set of concepts, R is a set of relations, I is a set of instances and A is a set of axioms [1]. This is also reflected in the most known ontology definition, which specifies ontology as an explicit and formal specification of a conceptualization of a domain of interest [1]. However ontology is something more than a set of concepts, relations, instances and axioms. In the previous paper it was noted that IF…THEN rules could be extracted from the ontologies as well [2]. Thereunto "net" OWL ontologies were conceived, t.i. ontologies without rule inserts in one of special rule languages [2]. The idea was to determine OWL ontology code fragments, which could be transformed to rules [2]. If the theoretic part of OWL ontology transformation to rules task was described in the previous research, this research is devoted to the practical part of this task.

The main purpose of this research is to describe an algorithm of OWL ontology transformation to rules. Seven basic cases were discovered when OWL code fragments could be transformed to rules [2]. The task is to process OWL ontology, i.e.., to look at each of the seven rule types and to supplement SWES concept map of these rules [3]. SWES concept map is the structure for storing generated rules in coded form. The second but not less important purpose of this research is to evaluate the efficiency of realized algorithm of OWL ontology transformation to rules. This algorithm is implemented in Java programming language, and it is important to find out how it applies to real-world software.

The final idea of the research is to develop such an expert system (SWES), which would be able with the help of OWL ontologies from the Web to extract rules and supplement its knowledge base in automatic mode, which is necessary for system functioning [3]. Thus, this algorithm of ontology processing and rule extraction from them is one of the key parts of the future expert system.

The paper is organized as follows. Section II gives an overview of the concept map structure. Section III describes OWL ontology transformation algorithm. The next section demonstrates performance of this algorithm. And section V gives conclusions and some ideas for future work.

II. Concept Map

The task of OWL code transformation to rules has to be changed to the task of OWL code transformation to a concept map and then to rules. In general, concept maps are graphical tools for organizing and representing knowledge [4]. Or in other words, concept maps are graphs, which include concepts as nodes and the relations between them as arcs [5]. However, in this paper concept maps are understood as specific data structures to store generated rules from OWL ontologies. Actually, concept maps are matrices - either on paper or in the realized program; that is, graphical representation is replaced with the matrix to facilitate processing with programs. Table I shows such a matrix. The form of transformation to a concept map and then to rules has several advantages. First of all, OWL transformation to a concept map allows to have such presentation of the rules, which can be coded to different rule formats, e.g., SWRL (Semantic Web Rule Language), Jena rules or other. The second advantage is that such presentation of rules in the form of concept map makes certain operations with rules more comfortable. For example, among these operations there may be different kinds of conflict resolution algorithms. Another advantage is that this sort of transformation to a concept map and then to rules allows to combine the Semantic Web Expert System with any inference engine more quickly, because in this case it would be necessary to transform the routine of rule coding to the required form of the selected inference engine only. It is obvious that in this case the structure of SWES becomes more flexible.

To develop the structure of the concept map, we should recall all seven kinds of OWL ontology transformations to rules [2]. In the first case, when class is not empty it can serve as a source for rule generation. For instance, if there is "Arrhythmia" class with three properties P1, P2, P3, then it becomes possible to generate the following rule [2]:

IF P1 **and** P2 **and** P3 **THEN** Arrhythmia (1)

In the second case there are two equivalent classes [2]. For example, there is "Arrhythmia" class and class "A" which is equivalent to class "Arrhythmia". In this case it becomes possible to generate the rule [2]:

IF A **equivalent** Arrhythmia **THEN** P1, P2, P3 ∈ A (2)

In case when there exists a relation between the two classes it is also possible to generate a rule [2]. For example, there are "Son" and "Father" classes and also the "hasParent" relation between these two classes. In this case it becomes possible to generate the rule [2]:

IF Son **THEN hasParent** Father, (3)

The rule means that if there is some instance of "Son", which belongs to class "Son" then this instance has relation "hasParent" to class "Father".

This happens when two concepts are defined and one of them is the part of the other [2]. For example, the class "Heart" is part of the class "Organism". It is possible to generate the following rules [2]:

IF Heart **and** "part of" **THEN** Organism, (4)

This rule means that if there is an instance of class "Heart" and the class "Heart" is the part of class "Organism" then this instance belongs to the class "Organism".

Another case when it becomes possible to generate a rule is the case when there are three classes "Heart", "Organism", "Blood" and the classes "Heart" and "Organism" are superclasses of the class "Blood", that is, there are "part of" relations between the class "Blood" and classes "Heart", "Organism". Hence, it is possible to generate the rule [2]:

IF Blood **and** "part of" (Heart **and** Organism)
THEN Heart **and** Organism, (5)

This rule means that if there is an instance of class "Blood" and the class "Blood" has the "part of" relations to classes "Heart" and "Organism" then this instance also belongs to classes "Heart" and "Organism".

There is yet another case where rules can be provided: when there is a class "Organism", which is the union of two classes - "Heart" and "Blood" (there are relations "part of" between the classes "Heart" and "Organism" as well as "Blood" and "Organism" [2]. The rule can be generated as follows:

IF (Heart **and**/**or** Blood) **and** "part of" Organism
THEN Organism, (6)

This rule means that if an instance belongs to the class "Heart" and/or "Blood" and "Heart", "Blood" classes are parts of "Organism" class, therefore this instance also belongs to the class "Organism".

If it is defined that one class does not belong to another, this can be transformed into a rule [2]. For example, there is class "Heart", which does not belong to class "Head". It becomes possible to generate the following rule:

IF Heart **THEN not Head**, (7)

This rule means, that if there is an instance of class "Heart" then it certainly does not belong to class "Head".

In order to keep these rules in the concept map, it is necessary to be able to code the following information:

- Components of the ontology;
- Types of the ontology components;
- Rules themselves.

The components of the ontology are their names. Types of the ontology components are one of the following identifiers:

- Rootclass;
- Subclass;
- Property;
- SameAs;
- Link;
- ComplementClass.

Ontology components and their types can be stored in two different arrays, where the elements have corresponding indexes.

In general, rules are two disjoint sets, where the first set is intended for storing rule conditions and the second set is intended for storing rule results. These two sets of rule conditions and rule results can be combined into one set, because they are disjoint. In addition, the maximum numbers of any rule conditions with rule results equal to the number of ontology concepts. Thus, each rule is a set of ontology components, where unused components have a "0" marker and used ontology components have the markers, which are divided in two categories:

- Markers of rule conditions,
- Markers of rule results.

Markers of rule conditions are the following:

- CAND – logical AND for conditions only;
- COR – logical OR for conditions only;
- CXOR – logical XOR for conditions only;
- CNOT – logical NOT for conditions only.

Markers of rule results are the following:

- RAND – logical AND for results only;
- ROR – logical OR for results only;
- RXOR – logical XOR for results only;
- RNOT – logical NOT for results only.

For example, there are seven components (concepts, links, properties – does not matter), which are obtained from one ontology. Hence, three rules are described like this:

TABLE I
Concept Map

	Component1	Component2	Component3	Component4	Component5	Component6	Component7
Rule1	CAND	0	CAND	0	0	RAND	0
Rule2	0	0	0	CAND	0	0	RAND
Rule3	0	CNOT	0	0	0	0	RNOT

Table 1 contains three coded rules (by rows):

Rule 1:

IF Component1 **AND** Component3 **THEN** Component6

Rule 2:

IF Component4 **THEN** Component7

Rule 3:
IF Component2 **THEN NOT** Component7

III. Transformation Algorithm

The proposed algorithm of OWL transformation to the concept map consists of 6 main steps and it is realized by using Jena Framework [6]. Each step of the algorithm means looking for a certain kind of rules. The first step is dedicated to the search for (4), (5), (6) kind of rules in the ontology. The second step is aimed at search for (1) kind of rules. The next step designed for extracting (2) kind of rules in the processed ontology. The fourth step is necessary for extracting (2) kind of rules, too. So is there, because there are 2 ways to encode equivalent concepts: using built-in "equivalent" identifier or by means of "sameAs" relation. The following step is necessary for extracting (3) kind of rules. The last step of this transformation algorithm is dedicated to the search for (7) kind of rules in the processed ontology. Each of these steps is designed as a separate Java function and during execution of these functions the algorithm analyzes ontology and fills in the concept map with rules in coded form as shown in previous section. It should be noted that each of these steps fills in its own auxiliary concept map, which is created prior to execution of each step, and which is copied to primary concept map in the end of the algorithm.

It is important to note that decomposition of this algorithm into 6 steps serves several purposes. First, this facilitates the process of programming. Indeed, it is much easier to look for errors in the parts of transformation algorithm than in the whole algorithm. Second, decomposition of transformation algorithm allows to evaluate the efficiency of this algorithm (this will be discussed in the next section), because decomposition allows evaluation not only of the whole algorithm, but also of parts of this algorithm. And third, the algorithm of ontology transformation to concept map is not necessary in itself. Primarily it is required to be implemented in SWES for processing OWL ontologies, extracting rules and supplementing SWES knowledge base with rules. In this connection such structural and functional decomposition of the algorithm allows to switch on or switch off the necessary functions according to the needs of SWES functional tasks.

Let us describe in detail all the 6 main steps of this ontology transformation algorithm to rules . The first step or function looks for all "part of" relations in the hierarchy of ontology classes and fills in the CM (concept map). The flowchart of the first function is shown here:

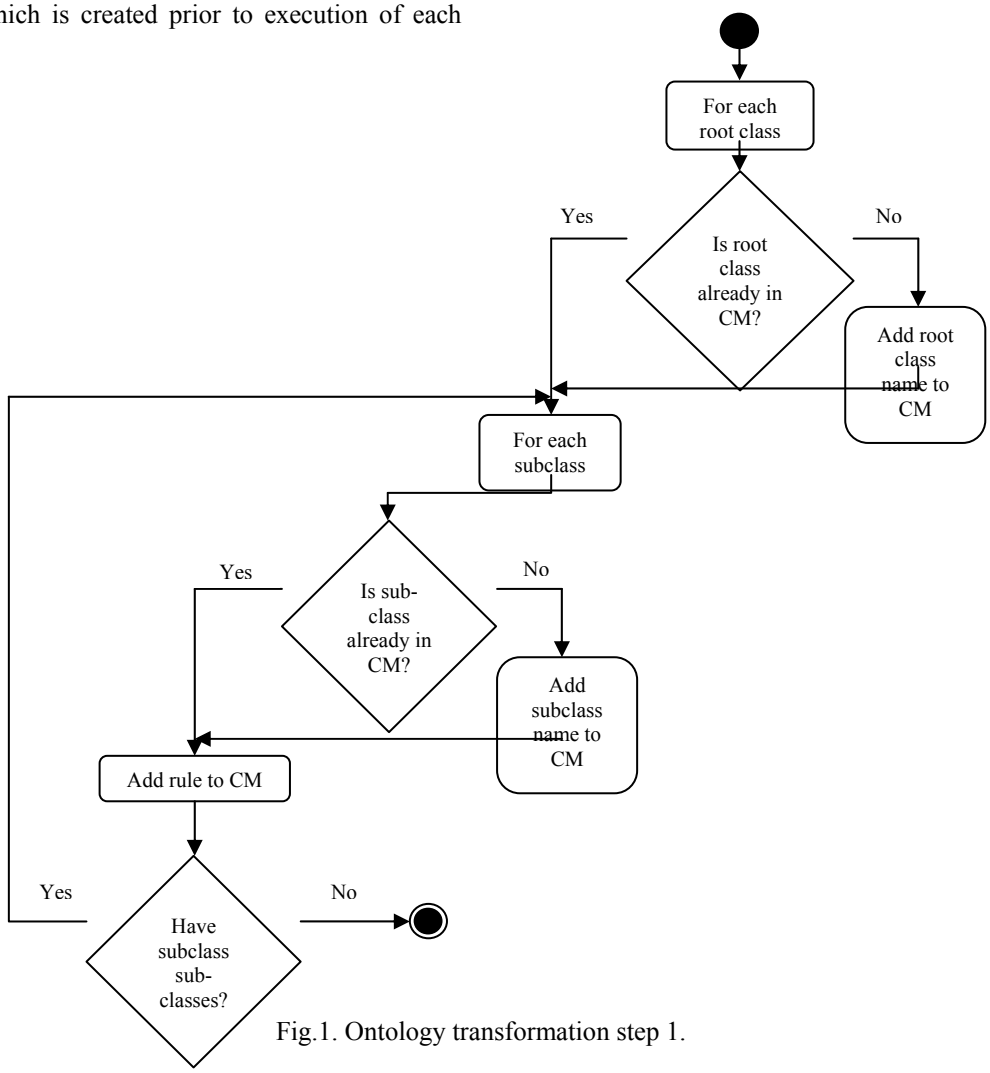

Fig.1. Ontology transformation step 1.

The second step or function looks for classes and their properties in the ontology to add "IF properties THEN class" kind of rule to the concept map. There are two concept maps here: the primary CM, which includes all rules, and an auxiliary CM, i.e. CM1, which contains "IF properties THEN class" rules only. Simple flowchart of the second function is shown here:

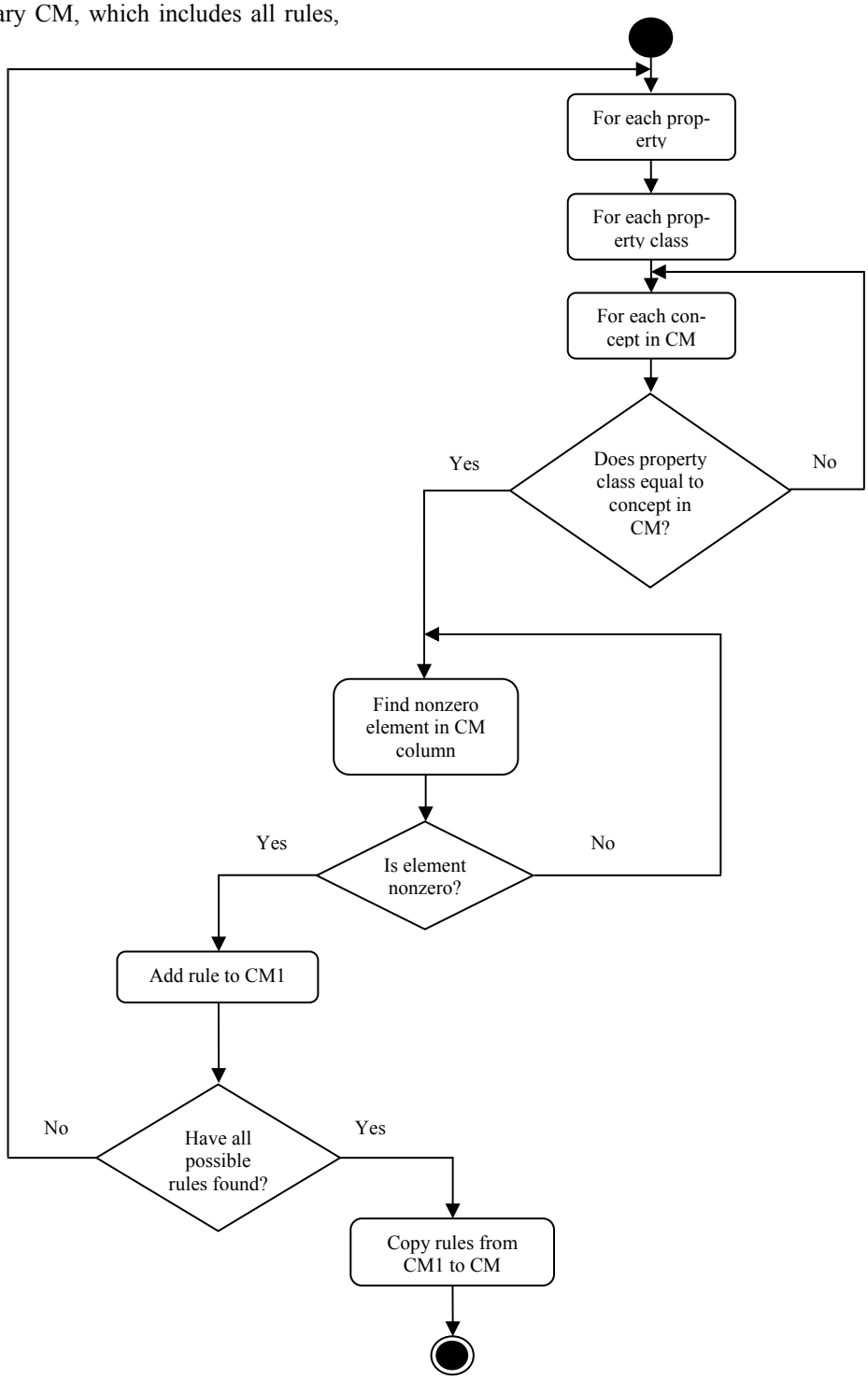

Fig.2. Ontology transformation step 2.

The third step looks for equivalent classes. It adds "IF equivalent classes THEN their properties are equivalent" kind of rule to the concept map. This step uses three concept maps. They are: the final concept map (CM), where all the rules are stored, the concept map 1 (CM1), for storing "IF properties THEN class" and concept map 2 (CM2), where rules from equivalent classes are kept. Simple flowchart of the third function is shown here on Fig. 3:

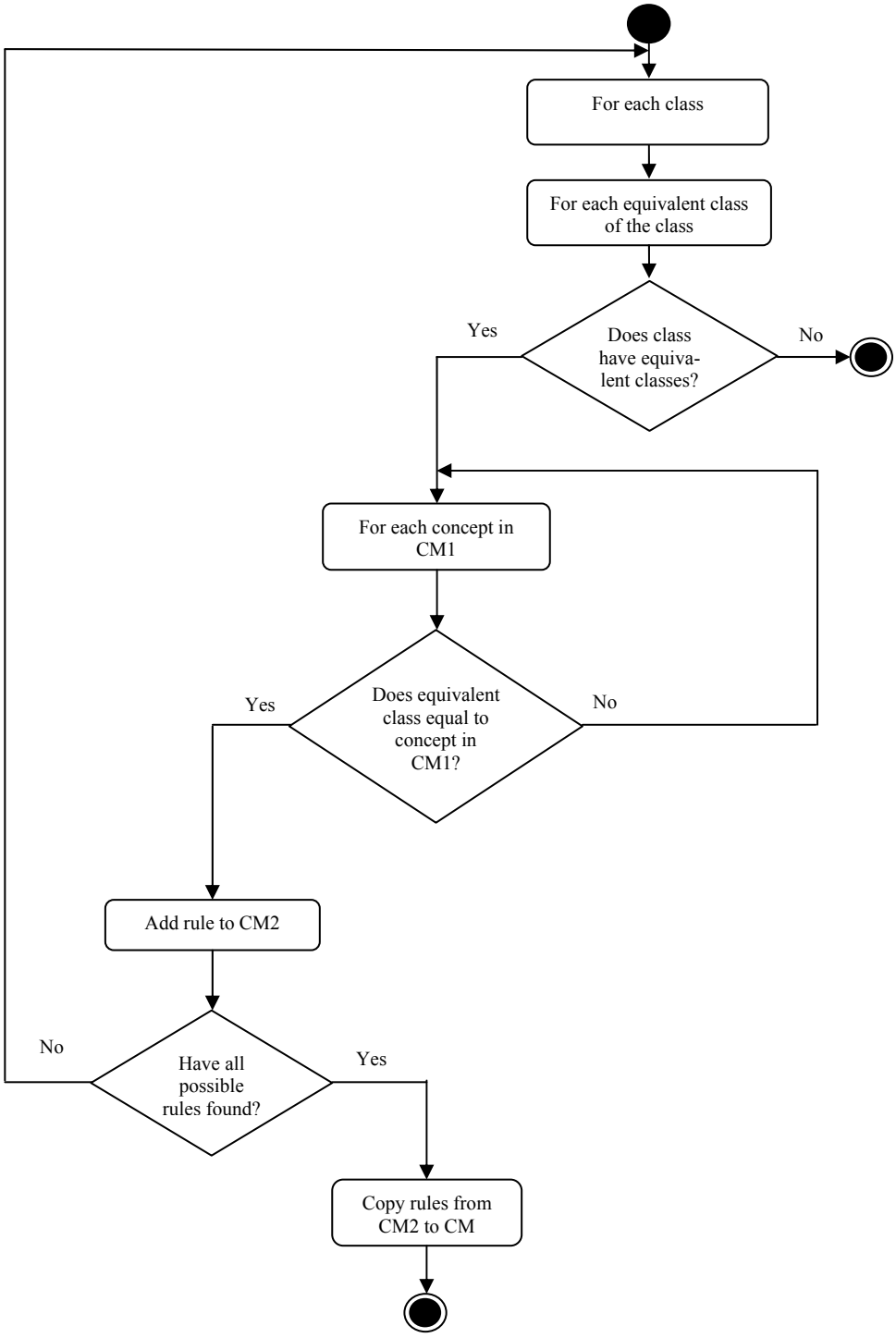

Fig.3. Ontology transformation step 3.

The fourth step or fourth function looks for "sameAs" classes in the ontology and adds "IF sameAs (equivalent) classes THEN their properties are sameAs (equivalent)" kind of rule to the concept map. The sub-algorithm of this step is identical to the sub-algorithm of the previous step; therefore, the flowchart of the fourth step is not discussed here.

The following step extracts from the ontology and adds to the concept map "**IF** there is an instance of one class, which has relation to another class **THEN** this instance has relation to this another class" kind of rule. This function needs two concept maps. The first concept map (CM) is designed for storing all rules, and the second concept map (CM1) is designed for storing "**IF** there is an instance of one class, which has relation to another class **THEN** this instance has relation to this another class" rules only. Simple flowchart of this function is shown here on Fig. 4:

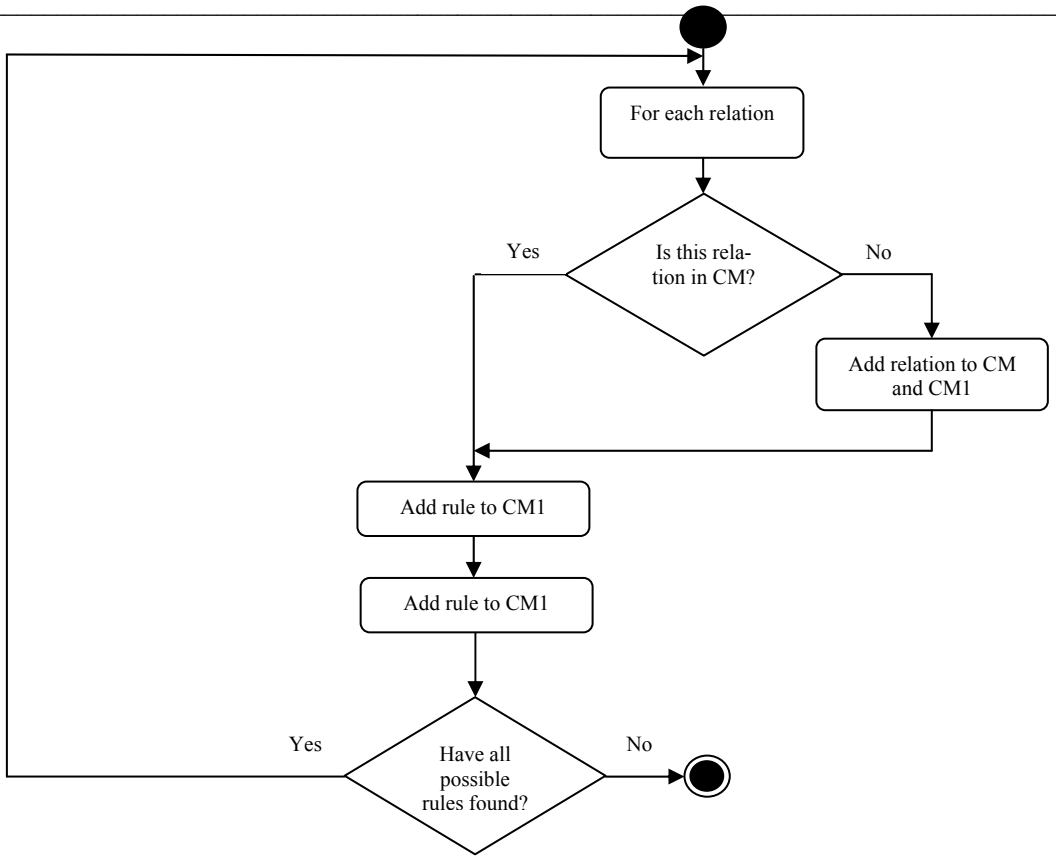

Fig.4. Ontology transformation step 5.

The last step of the transformation algorithm is dedicated to search for "**IF** class **THEN not** another class" kind of rules in the ontology. This step requires two concept maps: one for storing all rules and the other one for storing "**IF** class **THEN not** another class" kind of rules only. The flowchart of this step is very similar to that the previous step (except for the call of the function for searching complement classes instead of searching relations in the ontology) therefore, there is no need to show it separately.

Thus, as a result of the algorithm, which consists of 6 steps, concept map of the special form (Table 1) is shaped, where rules are coded and can be easily extracted to the form of "IF..THEN" or some special rule format.

IV. Performance of the Algorithm

Functionality of any algorithm is its key feature, which determines development of this algorithm. However, it is necessary to understand that performance of the algorithm is no less important feature of any algorithm than its functionality. If the functionality of the algorithm is what it capable of, then performance of the algorithm means how effectively it works. It happens so that inefficiency of the algorithm negates its functionality. Thus, functionality and performance of any algorithm are the two sides of one coin, and insufficiency of one leads to insufficiency in both.

Functionality of the described algorithm fully complies with the specs of the algorithm. That is, it complements the concept map with all defined kinds of rules. To properly evaluate performance of the algorithm it is necessary to develop the system of algorithm evaluation. Algorithm works with OWL ontologies, therefore it is necessary to utilize such ontologies, which would characterize performance of this transformation algorithm from different sides. Obviously, this can be done by means of ontologies with different number of classes (empty and not empty), object properties, equivalent and complement classes. It is thought that this will allow testing each step of the transformation algorithm.

The main indicator of algorithm performance, which we are interested in, of course is the time of execution. SWES will process real-world OWL ontologies, which will be found in the Web [3]. Generally, the size of such real-world ontologies is rather large because of large amounts of built-in knowledge (i.e.,. concepts, relations, instances) . In order to process a lot of large ontologies within a reasonable period of time it is necessary to provide the transformation algorithm with the adequate speed of execution. That is why we need to see how OWL ontology characteristics affect the speed at which the transformation algorithm is executed. For this purpose were developed 7 ontologies, by using Protégé 4.0.2.

TABLE II
Performance of OWL Ontology Transformation Algorithm

Nr p.k.	Ontology domain	Number of					Time, (ms)
		File size (kb)	Concepts	Relations	Datatype properties	Rules	
1	Transport	7	15	8	7	10	79
2	Software	12	23	1	4	23	125
3	Cinema	31	33	4	0	4	125
4	Robot world	46	119	149	0	120	156
5	Animals	53	53	2	5	52	188
6	Comparative Data Analysis	72	126	94	11	126	631
7	Economy	152	208	75	0	206	782

The ontologies, used for evaluation of the transformation algorithm, were found by means of SWOOGLE except for the first ontology. This first ontology was developed by the authors specially for our transformation algorithm testing purpose.

To evaluate the amount of time required for each step of the transformation algorithm functioning, the standard function in Java programming language is used, which returns for the user the current time in milliseconds: System.currentTimeMillis(). It is done as follows:

```
long start =  System.currentTimeMillis();
……
long end = System.currentTimeMillis();
long traceTime = (end - start);
```

It is important to note that this Java function, which returns the current time, has one serious drawback - it is the step, which is approximately equal to 14 milliseconds. It means that if the time of functioning is less than 14 milliseconds, the currentTimeMillis() function returns 0. This drawback was eliminated by using sufficiently large ontologies. Dependence of the ontology size on the runtime of the transformation algorithm is shown on Fig. 5:

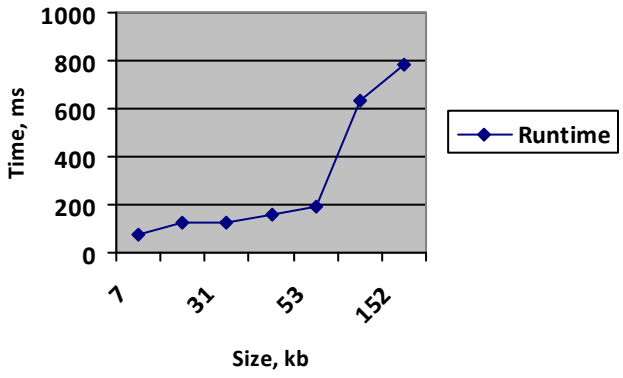

Fig. 5. Performance of the transformation algorithm.

Our OWL ontology transformation algorithm to rules was tested on the computer with the configuration of Pentium Dual Core 2.0 GHz and 2 Gb RAM. As seen in Figure 2, the larger is the ontology the more time is required for the transformation algorithm to extract rules. If for a 152 Kb sized ontology needs 782 milliseconds, then we can assume that 200 Kb sized ontology may require 1 second. It is known, however, that there is a lot of OWL ontologies in the Web, whose size exceeds 200 Kb. Processing of such ontologies will require the time longer than 1 second. In addition, it is assumed that SWES is not designed for just one OWL ontology processing. It is forecasted that SWES will usually process several OWL ontologies to complete one task. Obviously, this may take more time than it is permitted. Of course, computers develop and become more powerful, and this fact allows us to hope that soon this work will be carried out much faster. Nevertheless, ontology construction area has also shown some progress which is aimed at increasing the ontology size.

It is important to find the way how to reduce the impact of the problem of runtime, because the result of this research is a real system. Here we see several kinds of problem solutions:

- Using other programming languages (for example, C++) for time-critical code fragments of the transformation algorithm;
- Modification of SWES principle of operation ;
- Finding out what rules of the transformation algorithm are more useful in terms of SWES task performing.

The first problem solution is clear, therefore it is not discussed. The second solution means modification of how SWES functions. If the initial design supposed the SWES processes OWL ontologies using one of the known inference engines (for example, Pellet, Jena or others) and the transformation algorithm, which extracts all possible rules, then this second solution means processing OWL ontologies by using one of the known inference engines as before, but using OWL ontology transformation algorithm only if the result of inference engine exploitation is unsatisfactory. It allows saving computer resources. The third problem solution means that a special research to be carried out, which would focus on the two aspects: what are the rules of the transformation algorithm and how these rules influence achievement of a satisfactory result. According to this research it is possible to work out the strategy of SWES functioning. This strategy may be the following: first, use standard inference engine. If the inference engine does not

produce good results then to extract (1) kind of rule. If the result is still unsatisfactory, then, for example, (3) kind of rules in the ontology should be extracted, and so on.

V.CONCLUSION

As a result of this work OWL ontology transformation algorithm to rules (to the concept map, where rules are coded in a special form) was developed based on the ideas described in [2]. Several OWL ontologies were found in the Web to evaluate correctness and performance of our transformation algorithm. As a result of evaluation of the transformation algorithm it was concluded that some mechanism for reducing the algorithm runtime was required. Thus, the importance of this research was expressed by the fact that this research influenced the structure of SWES.

hree solutions of transformation algorithm big runtime were described. It was assumed that it was necessary to use just one of the solutions to tackle the problem of big runtime. However, now it seems that it is more advisable to try to use all of these solutions together. If optimization of program code by using C++ programming language is the task for the future, the rest of the solutions have to be exploited now. That is, our OWL ontology transformation algorithm has to be utilized only in case when use of standard inference engine is unproductive. In this case, additional rules, which can be obtained by means of proposed algorithm, are able to help SWES to cope with the task successfully. As regards the second proposal- investigate what rules, which can be obtained by OWL ontology transformation algorithm, should extract first of all, proves an important direction in future research. It is so because this proposal can increase SWES intellectual potential, add heuristics to the SWES task fulfillment. Therefore, this task is of high priority in future research towards the realization of new Semantic Web Expert System.

REFERENCES

[1] J. Davis, R. Studer, P. Warren, Semantic Web Technologies Trends and Research in Ontology-based Systems. Chichester: John Wiley & Sons Ltd, 2006.

[2] O. Verhodubs, J. Grundspņķis, Evolution of Ontology Potential for Generation of Rules, Craiova, in proceedings, 2012.

[3] O. Verhodubs, J. Grundspeņķis, Towards the Semantic Web Expert System, Riga: RTU Press, 2010.

[4] J. Novak, A. Canas, The Theory Underlying Concept Maps and How to Construct and Use Them. [Online]. http://cmap.ihmc.us/publications/researchpapers/theorycmaps/theoryunderlyingconceptmaps.htm. [Accessed: March 11, 2013]

[5] Anohina-Naumeca A., Graudiņa V., Grundspeņķis J. Using Concept Maps in Adaptive Knowledge Assessment // Advances in Information Systems Development "New Methods and Practice for the Networked Society". - Budapešta, Hungary: Springer, 2007. - pp 469-480.

[6] “Apache Jena”. [Online]. Available: http://jena.apache.org/. [Accessed: March 12, 2013]

Olegs Verhodubs got a bachelor degree in 2001 and got a master degree in computer sciences in 2004. Now he is engaged in expert knowledge representation and processing methods in intellectual training systems.

He worked as the engineer of computer technics for 4 years. Now he works in Riga Technical University as the assistant on scientific work. He is interested in artificial intelligence, expert systems and the semantic web.

Janis Grundspenkis graduated from Riga Polytechnic Institute (now Riga Technical University) in 1965. His specialty was electrical engineer of automation and telemechanics. He received his Dr.sc.ing. degree from Riga Polytechnic Institute in 1972 and his Dr.habil.sc.ing. degree in 1993 from Riga Technical University.

He is professor of systems theory at Riga Technical University. He is also a Dean of the Faculty of Computer Science and Information Technology, the Director of the Institute of Applied Computer Systems, and the head of the Department of System Theory and Design. His research interests are agent technologies, knowledge engineering and management, structural modeling for diagnostics of complex systems and development of intelligent tutoring systems.

He is member of Institute of Electrical and Electronics Engineers (IEEE), Association of Computer Machinery (ACM) and International Association for Development of the Information Society (IADIS). He is a full member of Latvian Academy of Science.