RUFRON: a Library for Generation of Rules from Ontology

Olegs Verhodubs

January 22, 2024

RUFRON: A library for Generation of Rules from Ontology

Olegs Verhodubs
oleg.verhodub@inbox.lv

Abstract. Ontology is a source to generate rules. Rules are one of the possible ways to display knowledge, and they are needed for knowledge-based systems and expert systems to function. The library Rufron is described in this paper, and it is designed as a library for programms to generate rules from the ontology and then to inference based on these rules.

Keywords: Ontology, Rules, Rules from Ontology, RUFRON

RUFRON available at: https://github.com/papillion80/RUFRON

I. Introduction

The quality of human life is increasing year by year, but this phenomenon has its costs. More and more trained specialists are needed to ensure life at the required level of quality. This applies to such different spheres of life as education, medicine, law, maintenance of various equipment - from household to industrial. Training professionals requires significant time and financial investment, so the idea of the ability to replicate the abilities of professionals in various fields has captured people for a very long time. With the advent of computers and Artificial Intelligence applications such as expert systems, this became possible. It could not be said that this became possible in all areas where we would like it, but in many very significant areas, it was possible. Advances in the development of expert systems were marked by such systems as MYCIN, INTERNIST, DENDRAL, XCON and many others [1]. Formally, an expert system is a computer system emulating the decision-making ability of a human expert [2]. But to put it simply, an expert system is a computer program that can completely or partially replace a human expert in a certain field. There are several classifications of expert systems, but classification according to the method of storing or in other words representing knowledge is fundamental. In turn, rule-based expert systems are expert systems in which the knowledge is represented by production rules. A production rule, or simply a rule, consists of an IF part (a condition or premise) and a THEN part (conclusion). IF condition THEN conclusion. Rules are a universal way of storing knowledge in the sense that they can be generated from various sources of information. The ability to generate rules from OWL (Web Ontology Language) ontologies [3], [4] is the most promising as the Web undergoes a transformation and the Web turns into the Semantic Web before our eyes. One of the facets of this transformation means filling the Web with ontologies. Ontologies is the way to describe the structure of some subject area without any restrictions that is why, having an ontology and the ability to generate rules from it, we immediately get the opportunity to extract the laws by which this subject area functions. Analyzing the trends in the development of the Web, it can be assumed that the ability to generate rules from an ever-increasing number of ontologies will be required by a large number of applications, so a tool is needed that will facilitate the development of future applications. This tool is a library with a set of functions that make it possible to generate rules from ontologies. This library was developed independently at my own expense and in my free time while working on the Semantic Web Expert System [5], now we are giving this library a name and making it freely available. The name of this library is RUFRON, which is short for RUlesFRomONtology.

The RUFRON library is available at https://github.com/papillion80/RUFRON

This paper is organized into several sections. The next section describes OWL constructs and types of rules that can be generated from these constructs. The third section presents the requirements and the architecture of the RUFRON library.

II. Theoretical basis for rule generation

OWL provides a means to describe ontologies. An ontology is a set of descriptions, where the main components are classes, properties, relations and individuals. More formally, an ontology is defined as a structure [6]:

$$O = <C, T, R, A, I, V>, \quad (1)$$

where there are ontology (O), concepts or classes (C), types (T), relations (R), attributes (A), instances (I), values (V).

Types, instances and values are not neccessary in our task that is why they will not be mentioned further. Our task is to describe a library that will make it possible to generate rules from the ontology and only classes, attributes and relations are required. This truncated set is enough to generate a decent number of rules that can be divided into several categories:

- identifying rules,
- descriptive rules.

Identifying rules distinguish an object from many other objects and descriptive rules connect new information with already known objects. In total, regardless of the category to which the rule belongs, there are 15 different types of rules described [3], [4], although there may be more types of rules. Here the initial version of the RUFRON programming library is described here, and it implements only four types of rules being generated from the OWL ontology. Let us list these types of rules:

1. identifying an object based on the known properties [3]. For example, there is a description of the class “Arrhythmia” with three properties P1, P2, P3 in the ontology:

```
<owl:Class rdf:ID="#Arrhythmia "/>
        <owl:DatatypeProperty rdf:ID="P1">
                <rdfs:domain rdf:resource="#Arrhythmia "/>
                <rdfs:range rdf:resource="xs:string"/>
        </owl:DatatypeProperty>
        <owl:DatatypeProperty rdf:ID="P2">
                <rdfs:domain rdf:resource="#Arrhythmia"/>
                <rdfs:range rdf:resource="xs:string"/>
        </owl:DatatypeProperty>
        <owl:DatatypeProperty rdf:ID="P3">
                <rdfs:domain rdf:resource="#Arrhythmia"/>
```

```
            <rdfs:range rdf:resource="xs:string"/>
        </owl:DatatypeProperty>
</owl:Class>
```

Visually it can be demonstrated like this (Fig.1.):

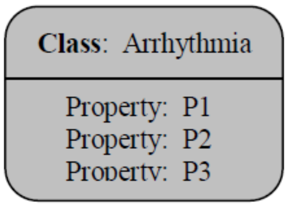

Fig.1. The “Arrhythmia” class with three properties.

It is possible to generate the following rule:

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

2. identifying an object based on class equivalence [3]. For example, there are two equivalent classes “Arrhythmia” and “A”:

```
<owl:Class rdf:ID="A">
        <owl:equivalentClass>
                <owl:Class rdf:ID="Arrhythmia"/>
        </owl:equivalentClass>
</owl:Class>
-------------------------- OR -----------------------
<owl:Class rdf:ID="A">
        <owl:sameAs rdf:resource="# Arrhythmia"/>
</owl:Class>
```

Visually it can be demonstrated like this (Fig.2.):

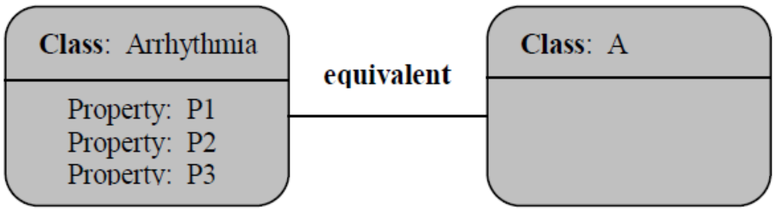

Fig.2. Two equivalent classes “Arrhythmia” and “A”.

It is possible to generate the following rules:

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

IF P1 **and** P2 **and** P3 **THEN** A (3)

3. complemention with data, exploiting relation. For example, there are “Son” and “Father” classes and also “hasParent” relation between these two classes:

```
<owl:ObjectProperty rdf:ID="hasParent">
        <rdfs:domain rdf:resource="#Son"/>
        <rdfs:range rdf:resource="#Father"/>
</owl:ObjectProperty>
```

Visually it can be demonstrated like this (Fig.3.):

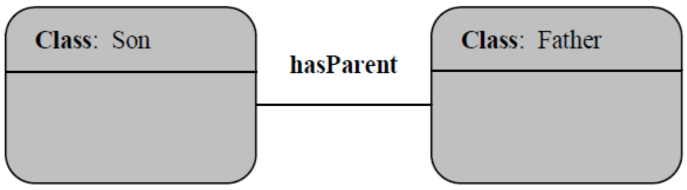

Fig.3. Two classes and a relation.

It is possible to generate the following rules:

IF Son **THEN** **hasParent** Father (4)

4. object identification based on part-whole relation. For example, there is a class “Heart” that is the part of class “Organism” (Fig.4):

```
<owl:Class rdf:ID="Heart">
        <rdfs:subClassOf rdf:resource="#Organism"/>
</owl:Class>
--------------------- OR ----------------------
<owl:Class rdf:ID="Heart">
        <rdfs:subClassOf>
                <owl:Class rdf:ID="Organism"/>
        </rdfs:subClassOf>
</owl:Class>
----------------------- OR ----------------------
<owl:Class rdf:ID="Heart">
        <rdfs:subClassOf>
                <owl:Class rdf:about="#Organism"/>
        </rdfs:subClassOf>
</owl:Class>
```

Visually it can be demonstrated like this (Fig.4.):

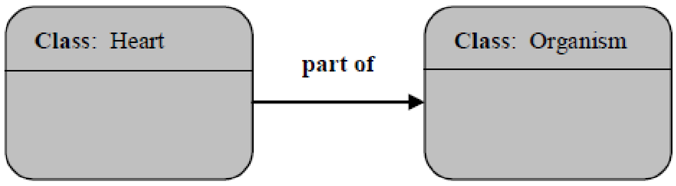

Fig. 4. Class “Heart” is part of class “Organism”.

It is possible to generate the following rule:

IF Heart **and** “part of” **THEN** Organism (5)

5. not belonging to the object. For example, there is a class “Heart” and this class does not belong to class “Head”:

```
<owl:Class rdf:ID="Head"/>
<owl:Class rdf:ID="Heart">
        <owl:complementOf rdf:resource="#Head"/>
</owl:Class>
```

Visually it can be demonstrated like this (Fig.5.):

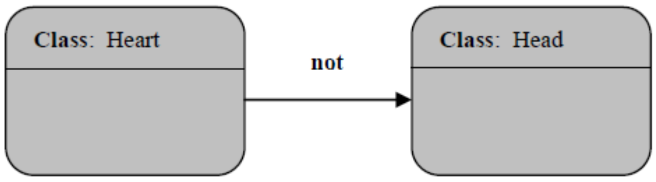

Fig. 5. Class “Heart” is not class “Head”.

It is possible to generate the following rule:

IF Heart **THEN** ***not*** Head (6)

The mentioned types of rules are sufficient for stable logical reasoning. Now the architecture of the library is being described.

III. Architecture

The RUFRON is a library for generation of rules from the ontology. Rules from an ontology are being generated by searching for typical code fragments in the OWL ontology and subsequent formation of a rule in the form IF... THEN... in accordance with the data from the OWL code fragments. Processing of code fragments in OWL is carried out using Apache Jena framework [7] and Java programming language. Thus, RUFRON needs Java and Apache Jena.

RUFRON consists of three classes: Matrix, SetMatrix and Rufron. The class Matrix is designed to work with a matrix. It has the low-level methods to add rows and columns to the matrix, to set some value to the certain cell of the matrix and also several methods to get concept name, type and location based on some property. The work with a matrix is necessary, because generated rules are stored in the form of matrix. The Matrix class has the following methods (Table I):

TABLE I. Methods of class Matrix.

Method specification	Purpose
`public String get(int i, int j)`	gets relation name
`public void set(int i, int j, String value)`	sets relation name
`void addRowCol(String concept, String concType)`	adds new relation and its type
`int getConceptNumber(String concName)`	returns concept index
`int getConceptNumberLast(String concName)`	gets concept index
`String getConceptName(int i)`	returns concept name
`String getConceptType(int i)`	returns concept type
`void setConceptType(int i, String concType)`	sets concept type
`int getLinkNumber(int i)`	returns relation quantity of concept

`void refine()`	corrects names (delete indexes to concepts and _)
`void delete(int ind)`	deletes concept by index
`int duplicate(String concept_name)`	detects if concept is added to matrix

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 [8]:

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

Markers of rule conditions are the following: CAND/ COR/ CXOR/ CNOT – logical AND/ OR/ XOR/ NOT for conditions; markers of rule results are the following: RAND/ ROR/ RXOR/ RNOT – logical AND/ OR/ XOR/ NOT for results. [8]

For example, there are seven components (concepts, links, properties), which are obtained from one ontology. Hence, three rules are described (Table II):

TABLE II. Matrix of rules.

	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

The following rules are code in the Table II:

IF Component1 **AND** Component3 **THEN** Component6 (7)

IF Component4 **THEN** Component7 (8)

IF Component2 **THEN NOT** Component7 (9)

The class SetMatrix is designed to fill the matrix with ontology data: classes, properties, links. This class realises such methods (Table III):

TABLE III. Methods of class SetMatrix.

Method specification	Purpose
`Matrix getRootClass(OntModel mod, Matrix matrix)`	finds a root class
`Iterator getAllClasses(OntModel mod)`	iterator of classes
`void getSubClass(OntClass rootcls, Matrix matrix)`	finds all sub classes
`void getSynonClass1(OntModel mod, Matrix matrix, Matrix  matrix1, Matrix matrix2)`	finds equivalent classes
`void getSameAsClass1(OntModel mod, Matrix matrix, Matrix matrix1, Matrix matrix2)`	finds equivalent classes
`void getObjectProperty1(OntModel mod, Matrix matrix, Matrix matrix1)`	finds relations between classes
`void getDatatypeProperty1(OntModel mod, Matrix matrix, Matrix matrix1)`	finds class attributes
`void getComplementClass(OntModel mod, Matrix matrix, Matrix matrix1)`	finds nonequivalent classes

The class Rufron is the main class in this programming library. It is this class that must be connected to the program that will generate rules from the OWL ontology, and the classes Matrix and SetMatrix are auxiliary. The class Rufron is designed to connect ontology and also to generate all types of rules from the ontology in its own matrix. For now, the class Rufron consists of only one method:

```
void generateRulesFromOntology(String ont)            (10)
```

This method takes the ontology name and location (String ont) and generates all rules into matrices that are declared inside this method. This is not the ideal construction, however, there is no need to use the programming library on an industrial scale. Now let us describe the perspective of the library RUFRON.

IV. Future work

The described library had been developed during work on the Semantic Web Expert System [5]. The Semantic Web Expert System is an expert system that uses the ontologies from the Web to generate rules and to fill the knowledge base with generated rules. Then, a user can exploit the Semantic Web Expert System, filled with generated rules, to obtain an expert opinion on an issue of interest. If you use this library only for this purpose (Semantic Web Expert System), then even in this case it can be improved. There are a lot more types of rules that can be generated from the ontology than realized in Rufron. At this moment only types of rules published in [3] are realized, but there also are other types of rules being generated from ontology that was pullished in [4] and have not realized in code yet. And these are not all possible improvements. For example, it is possible it would be more effective to store generated rules in other data structures (not in matrices) and these data structures can situate in other places. Further, it would be possible to untie this programming library from Apache Jena, thus providing this library with another degree of freedom.

It is much more interesting to consider promising growth points for this library. Most of the growth points being viewed are related to the possible expansion of this library into other promising projects. The first point of growth is associated with the promising ability to output rules in various syntactic formats. For example, the classic rule format in the form of “IF..THEN” can be replaced with the rule format used in Apache Jena [9]:

```
[rule1: (?a eg:p ?b) (?b eg:p ?c) -&gt; (?a eg:p ?c)]            (11)
```

Of course, in the future, other formats for presenting rules may be needed, too. And this has to be reflected in RUFRON.

The second groth point is associated with the ability to generate rules not only from ontology, but also from other sources. It is already clear that rules can be generated from a relational database structure [10]. An even more exciting prospect is the ability to generate rules from raw text [11], [12]. That is not all: scientific intuition suggests that rules can crystallize from other information sources. And all these potential possibilities should, of course, be implemented in the Rufron.

Acknowledgments

This paper and the code were created solely at the expense of the author and using the author’s computer.

References

[1] Peter J.F. Lucas & Linda C. van der Gaag, Principles of Expert Systems, Amsterdam, 1991

[2] Jackson Peter, Introduction To Expert Systems (3 ed.). Addison Wesley, 1998

[3] Verhodubs Olegs, Evolution of ontology potential for generation of rules, 2012.

[4] Verhodubs Olegs, Ontology as a source for rule generation, 2014.

[5] Verhodubs Olegs, Towards the Semantic Web Expert System, 2011.

[6] Davies John, Struder Rudi, Warren Paul, Semantic Web Technologies, 2006.

[7] Apache Jena. Available online: jena.apache.org

[8] Verhodubs Olegs, Grundspenkis Janis, Algorithm of Ontology Transformation to Concept Map for Usage in Semantic Web Expert System, 2013

[9] Reasoners and rule engines: Jena inference support. Available online: https://jena.apache.org/documentation/inference/#reasonerAPI

[10] Verhodubs Olegs, Generation of Rules from the Relational Database Structure, 2023.

[11] Verhodubs Olegs, Experiments of Rule Extraction from Raw Text, 2021.

[12] Verhodubs Olegs, Rule Mining from Raw Text, 2021.