ONTOP: A COMPONENT FOR ACQUIRING INFORMATION FROM OWL ONTOLOGIES

Ines ČEH*, Milan ZORMAN**, Matej ČREPINŠEK*, Tomaž KOSAR*, Marjan MERNIK*, Jaroslav PORUBÄN***
*Programming Methodologies Laboratory, Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia, e-mail: {ines.ceh, matej.crepinsek, tomaz.kosar, marjan.mernik}@uni-mb.si
**Laboratory for System Design, Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia, e-mail: milan.zorman@uni-mb.si
***Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, Slovak Republic, tel.: +421 55 602 2565, e-mail: jaroslav.poruban@tuke.sk

ABSTRACT

This paper presents a parser for OWL DL. OWL DL, a sub-language of OWL, allows efficient reasoning with computability assurance. The OWL parser, named OntoP, has been developed specifically for the requirements of a larger framework: Ontology2DSL. Ontology2DSL enables the semi-automated construction of a formal grammar, as well as programs incorporating domain-specific languages (DSLs) from OWL ontologies. In this presentation of the parser, we focus on the data structure that the parser uses to store the extracted information from an OWL document (written in RDF/XML syntax), the algorithm used to construct and visualise the class hierarchy of the ontology, a converter that can transform the RDF/XML to Manchester OWL syntax, and those ontology metrics that highlight key ontology characteristics.

Keywords: *parsing, ontology, OWL, RDF/XML syntax, Manchester OWL syntax*

1. INTRODUCTION

Web Ontology Language (OWL) is a semantic markup language developed for the representation of information on the semantic web [1], [2]. It was designed by the World Wide Web Consortium (W3C). It reached the status of a recommendation in 2004. OWL ontologies are used for modeling domain knowledge. Ontology describes the concepts in a domain and the relationships among those concepts [3]. A review of existing literature provides several definitions for ontology. The more widely-accepted is the definition by Studer et al. that defines ontology as: "An ontology is a formal, explicit specification of a shared conceptualization." [4], [5]. Besides its formal W3C recommendation status, OWL is a success because of the vast set of tools that enable working with OWL ontologies. These tools enable the creating and editing of OWL ontologies (Protégé [6], Swoop [7]), and offer inference and reasoning (Pellet [8], Fact++ [9]). In addition to these tools, there are also various Application Programming Interfaces (APIs) that enable the use of ontologies within various applications (OWL-API [10] and Jena [11]).

OntoP the presented parser, as explained in further detail in Section 3, was developed for the Ontology2DSL framework [12], [13]. The Ontology2DSL framework, shown as a workflow diagram in Fig. 1, enables the semi-automated construction of formal grammars [14], DSLs [15], and programs, all from an OWL ontology. The framework accepts an OWL document (written in RDF/XML syntax) as its input. It then proceeds with parsing the document and uses the acquired information to fill its internal data structure, known as an ontology data structure (ODS). The transformation pattern (TP), a sequence of rules that construct the grammar and programs, is run over ODS. Some of the rules in the TP require some involvement and activity from a DSL engineer. The final results of the framework are the DSL grammars and programs.

It was decided to construct DSLs from OWL ontologies because OWL is currently the most widely-used ontology language [16]. RDF/XML syntax was selected because it is the default syntax of OWL and, therefore, must be supported by all OWL-compatible tools [1], [2]. The choice of language and syntax was made so that it would consequently support the widest possible range of existing ontologies for the construction of DSLs.

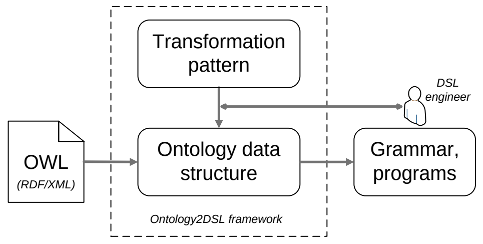

Fig. 1 Ontology2DSL framework workflow diagram

The OntoP parser is one of the five main components of the Ontology2DSL framework. The architecture and major components of the framework are presented in more detail in [12]. The tasks of the OntoP parser within the process of constructing the language grammar and programs, include the following:

- Creating and filling the internal data structure, ODS.
- Creating and visualising the ontology class hierarchy, property hierarchy (object and datatype properties), and the list of individuals.
- Acquiring and presenting all available information for each of the ontology components (classes, object properties, datatype properties, and individuals), and the ontology itself.
- Acquiring and representing information on key ontology characteristics.

Unauthenticated | 194.138.39.60
Download Date | 1/15/14 3:04 AM

The construction of language grammar and programs from an OWL ontology is executed over three phases. During each phase OntoP performs one or more of the tasks listed previously. The phases are:

- **Ontology selection.** As stated Ontology2DSL enables the construction of DSL grammars and programs from OWL ontologies. The development of a DSL is significanty shortened if it is constructed from an existing ontology. When a DSL engineer wishes to use an existing ontology for DSL construction he has to consider whether the ontology is appropriate for the task. Key ontology information is vital for evaluating the ontology. For instance, if an ontology only contains classes, and there are only a few of them, it is highly probable that such an ontology contains an insufficient number of concepts needed for DSL development. Based on information from key ontology characteristics, a DSL engineer can quickly determine whether the ontology is worth considering. The condition for the DSL development is a good understanding of the ontology semantics. The engineer must determine the purpose for which the ontology was created, what concepts it describes, what is the overlap of these concepts with the DSL needs, etc.
  During this phase OntoP acquires and presents information to the DSL engineer on key ontology characteristics. It also displays all the hierarchy, as well as a list of individuals and information on all ontology parts.

- **Transformation.** The transformation is done over four steps using Ontology2DSL. During the first step, the user selects the ontology over which the transformation will run. In the second step, the user select the pattern that will be used in the transformation. Since the transformation does not usually include the entire ontology DSL requirements and does not fully match with all of the ontology concepts, in step three the DSL engineer excludes from the class hierarchy all those classes unused during transformation. During step four, the TP is run over the ODS. Step four is comprised of a variable number of sub-steps. This number depends on the number of rules the individual TP holds.
  During this phase OntoP creates and fills ODS and enables the DSL engineer to view the ontology class hierarchy and information on individual classes.

- **Inspection and verification of created grammar and programs.** The DSL engineer inspects the newly-created grammar and programs for any inconsistencies or errors. If any are found the engineer has the option of directly editing the grammar, altering the transformation pattern, or the source ontology. If the DSL engineer changes a transformation pattern it is necessary to re-run phases two and three. During this the results from phase one are used. Phases two and three have to be run from the beginning. If the DSL engineer uses a different ontology all three phases need to be executed again. If no inconsistencies or errors are detected, the process is complete.
  For the DSL engineer it is useful to re-analyze the ontology characteristics at this time. For example, based on the information regarding the number of classes, properties, and individuals, the DSL engineer will be able to create an image of the production count, types of grammar, and programs that could be developed from ontology using such characteristics. This will help the DSL engineer with phase one, the ontology selection, and when working on new projects.

This paper presents the OntoP parser, the data structure that the parser uses to store the extracted information from an OWL document (written in RDF/XML syntax), the algorithm used for constructing and visualising the class hierarchy of the ontology, a converter that can transform the RDF/XML to Manchester OWL syntax, and those ontology metrics that highlight key ontology characteristics (number of classes, properties, individuals, etc.).

The organization of the paper is as follows: Section 2 presents the ontology web language OWL. Section 3 introduces the ODS, an algorithm for constructing and visualising the class hierarchy, a RDF/XML-Manchester OWL syntax converter, and ontology metrics. The conclusion and ideas for future work are summarized within Section 4.

2. THE WEB ONTOLOGY LANGUAGE OWL

2.1. OWL Sublanguages

The original OWL specification includes the definitions of OWL's three sub-languages, having different levels of expressiveness. These languages are OWL Full, OWL DL and OWL Lite (ordered by decreasing expressiveness) [1], [2], [17]. OWL Full is not a sub-language of the OWL language, it is the full OWL language itself. OWL Full allows the user to ”say anything about anything”. The flexibility of OWL Full is detrimental to computational efficiency. OWL Full is not decidable, this means that there are no known algorithms that are capable of assuring complete inference for a given OWL Full ontology. The ”DL” in the name of OWL DL sub-language stands for Description Logic, an important subset of first-order logic [18]. OWL DL uses the same constructs as OWL Full. Usage regarding some of these constructs is restricted [19]. These restrictions make OWL DL decidable and, therefore, an algorithm exists that ensures complete inference for any OWL DL ontology. Decidability, however, makes no statements on the efficiency of the algorithm and does not guarantee real-time completion. OWL Lite is essentially OWL DL with a sub-set of its language constructs.

2.2. OWL DL

The three basic components of the OWL DL sub-language, which during the continuation of this paper is only referred to as OWL, are: classes, properties, and individuals [1], [2], [3].

Classes are interpreted as those sets containing individuals [3]. OWL defines two types of classes: simple named and predefined classes. Simple named are those classes defined by the user. The predefined classes, as provided by

Unauthenticated | 194.138.39.60
Download Date | 1/15/14 3:04 AM
VERSITA | EMERGING SCIENCE PUBLISHERS

OWL, are ”Thing” and ”Nothing.” Whilst ”Thing” is the superclass of all classes, ”Nothing” is an empty class and can be a subclass of every class. Two types of simple named classes are defined; primitive and defined classes. Whilst the primitive classes are described only with necessary conditions, the defined classes are described with at least one set of necessary and sufficient conditions. If a class is only described using necessary conditions, then it can be ascertained that if an individual is a member of that class it must satisfy the conditions that describe this class. If a random individual satisfies the conditions, it cannot be stated that it must be a member of this class. If the class should later be defined using necessary as well as sufficient conditions, it would be possible to make decisions using both statements. Classes can be disjoint. Classes can be organized into hierarchies. The second component is properties, which is a binary relation. The two main types of properties in OWL are object properties and datatype properties. Object properties link objects with other objects, and datatype properties with data values. OWL uses data values defined within the XML Schema Definition Language (XSD) (version 1.0) [1]. Properties, similarly to classes, are also organized into hierarchies. Every property has a domain and range. A property links domain individuals with range individuals. Object properties can also have inverse properties and in that case domain and range are swapped. Individual properties can have multiple domains and ranges. OWL allows properties to have special characteristics. Consequently, objcet properties can be transitive, symmetrical, functional and inverse functional, datatype properties can only be functional. The third component is the set of individuals, which are members of user-defined classes. A document is an OWL DL if its corresponding graph conforms to the rules for OWL-DL [20].

2.3. Syntaxes

Syntax is a set of rules that defines a language’s format. The OWL W3C recommendation defines the standard exchange syntax as the RDF/XML syntax [1], [2], [18]. This is the syntax that all OWL-compatible tools should support. Other syntax also exists in addition to RDF/XML syntax. Some examples include: Turtle, Abstract Syntax [1], [2], Manchester OWL [21], and others. Different syntax is optimised for different purposes. For our work, the most important syntaxes are the RDF/XML and the Manchester OWL syntax.

RDF/OWL, the primary syntax that all OWL-compatible tools must support, is the XML syntax intended to represent RDF triples. RDF/XML is a very extensive syntax. Most tools use this syntax as the default for saving OWL ontologies. The advantage of this syntax is that it is widely-supported but its disadvantage is that it is very extensive and difficult for humans to comprehend.

Manchester OWL, the compact-text syntax, is derived from Abstract Syntax. When compared to it, Manchester OWL is less extensive and minimizes the use of brackets. This results in it being easy to read, write and edit. Relatively difficult expressions written in this syntax should be as easily readable as regular English language. Syntax uses natural language words such as ”AND”, ”SOME” and ”NOT” instead of mathematical symbols. Reading and understanding is also made easier with the infix notation. The simplicity of this syntax is its major advantage but its disadvantage is that it is very clumsy regarding some OWL axioms. How compact and readable the Manchester syntax is when compared to XML/RDF syntax is represented in Fig. 2 and 3. Both show the definition of VegetarianPizza class, in Fig. 2 it is written in RDF/XML while in Fig. 3 it is written in Manchester OWL syntax.

```
<owl:Class rdf:about="#VegetarianPizza">
<owl:equivalentClass>
  <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
     <rdf:Description rdf:about="#Pizza"/>
     <owl:Restriction>
       <owl:onProperty rdf:resource="#hasTopping"/>
        <owl:allValuesFrom>
         <owl:Class>
           <owl:unionOf rdf:parseType="Collection">
             <rdf:Description rdf:about="#CheeseTopping"/>
             <rdf:Description rdf:about="#VegetableTopping"/>
           </owl:unionOf>
         </owl:Class>
       </owl:allValuesFrom>
     </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>
</owl:equivalentClass>
</owl:Class>
```

Fig. 2 Definition of VegetarianPizza class written in RDF/XML syntax

```
Class:
VegetarianPizza

Equivalent classes:
Pizza and hasTopping only (CheeseTopping or VegetableTopping)
```

Fig. 3 Definition of VegetarianPizza class written in Manchester OWL syntax

3. ONTOP PARSER

Parsing is the procedure during which the parser provides a concrete representation of a document containing an ontology, and then builds an internal representation compliant with the document. OntoP is a parser implemented in the programming language C# which is part of the .NET framework. OntoP uses the XML document object model (DOM) and those XML parsers associated with it, in order to parse RDF/XML documents. It is intended to parse OWL ontologies. OntoP supports all syntactic elements of OWL language. Elements of the language, their descriptions and examples can be found in [1], [2], [19], [22].

3.1. Ontology data structure

The Ontology2DSL framework needs the ontology to be in proper format. For this purpose OntoP takes a concrete representation of an OWL document written in RDF/XML syntax and constructs the inner representation (fully compliant with the OWL document). The inner representation is the ontology data structure ODS, to which OntoP writes the extracted information. ODS is used to run transformation patterns over.

Unauthenticated | 194.138.39.60
Download Date | 1/15/14 3:04 AM
VERSITA EMERGING SCIENCE PUBLISHERS

ODS is composed of the following data structures: tree of classes, tree of object properties, tree of datatype properties, and a list of individuals. The parser fills the ODS in the same sequence as listed here. The nodes in the individual trees are objects that contain information about the individual ontology blocks stored within each node. Each node stores the name of the block, as well as the following:

- equivalent classes, superclasses, members, disjoint classes, comments, and labels (class tree),
- characteristics, domains, ranges, inverse properties, and super properties (object property tree),
- characteristics, domains, ranges, and super properties (datatype property tree), and
- types, data property assertion and object property assertions (list of individuals).

3.2. Hierarchy construction and visualization algorithm

During the process of ontology analysis, the DSL engineer determines which concepts (classes) the ontology contains and how they overlap the concepts of the DSL being developed. Experience shows that ontology often contains concepts that are irrelevant and these concepts should be removed before the transformation. In order to ease the elimination process for the DSL engineer, it was decided to enable a visual representation of the ontology class hierarchy. This ontology hierarchy is visualised in the form of a tree in which each node is an ontology class. Using visual representation, the DSL engineer can easily deselect those classes that are unrequired during the transformation, and therefore removing the long procedure of eliminating unnecessary classes from the RDF/XML syntax. A visual representation of the ontology consequently diminishes the time needed for DSL development.

The OntoP parser uses a two-part algorithm for constructing the class hierarchy. The first part of the algorithm creates lists of classes from an OWL file, in RDF/XML syntax. Individual lists contain elements from a branch derived from each of the ontology's top-classes (a branch lists all subclasses of a top class; the number of branches is limited with the number of top classes multiplied with the number of sub-classes). These lists are in ascending order from the 0 index as a leaf of the branch (the lowest subclass), to the higher indexes as branches containing higher-classes. During the second part of the algorithm, the tree of classes is filled with these lists. The algorithm performs the following steps:

- In step one it loads an OWL document and removes all non-essential information. Non-essential information includes user comments written in the XML file that do not syntactically match the ontology comments.
- In the second step, the algorithm acquires a list of all ontology classes. For each of them, the parent class is found and stores them in the form of "ClassName-ParentName" pairs. The first element is the class, and the second is its parent class. Each pair is added to a list of pairs. A part of the list of pairs for Pizza ontology is presented in Fig. 4.
- In the third step, the algorithm loops over the list of pairs and performs data cleaning and verification procedures.
- In step four, the remaining pairs are collated into lists. This procedure starts with the first pair, then scans the remaining pairs to find the pair where the parent class of the first pair is found to be the class name. From that pair, the parent name can be extracted. Therefore, a first list can be formed with three members. The procedure is repeated until no more parents can be extracted. Then the entire procedure is repeated for all remaining pairs until all of them have been transformed onto hierarchy lists. The merging procedure (pairs to lists) is presented in Fig. 5. A part of the list of all merged lists is presented in Fig. 6.
- Finally, in the fifth step the algorithm fills a tree by first creating the root of the tree called "Thing." ("Thing" being the superclass of all ontology classes.) The lists are then ordered according to length. The longest lists are ranked highest, therefore their elements are the first to be written to the tree. A part of the lists is shown in Fig. 6. The number of elements in the longest list provides the depth of the tree. Each list is then processed individually. Each list is first checked for length and, if the list is amongst the longest, the final element from it will become a child-node of the root-node ("Thing"), unless that node has been inserted already. The other elements from the list become its children (each a level lower). The procedure is run recursively for all the lists.

For the construction of the hierarchy of objects and datatype properties, the same algorithm is used as for the construction of the ontology class hierarchy.

```
PizzaTopping ⟶ »«
JalapenoPepperTopping ⟶ PepperTopping
PepperTopping ⟶ VegetableTopping
RedPepperTopping ⟶ PepperTopping
VegetableTopping ⟶ PizzaTopping
GreenPepperTopping ⟶ PepperTopping
Thing ⟶ »«
. . .
```

Fig. 4 An excerpt of the list of pairs for the Pizza ontology

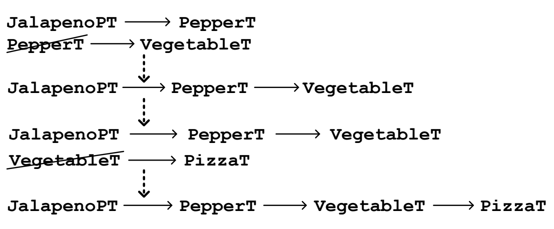
JalapenoPT ⟶ PepperT
~~PepperT~~ ⟶ VegetableT
JalapenoPT ⟶ PepperT ⟶ VegetableT
JalapenoPT ⟶ PepperT ⟶ VegetableT
~~VegetableT~~ ⟶ PizzaT
JalapenoPT ⟶ PepperT ⟶ VegetableT ⟶ PizzaT

Fig. 5 The merging procedure

Unauthenticated | 194.138.39.60
Download Date | 1/15/14 3:04 AM

$$\begin{tikzcd}
\texttt{JalapenoPT} \arrow[r] & \texttt{PepperT} \arrow[r] & \texttt{VegetableT} \arrow[r] & \texttt{PizzaT} \\
\texttt{GreenPT} \arrow[r] & \texttt{PepperT} \arrow[r] & \texttt{VegetableT} \arrow[r] & \texttt{PizzaT} \\
\texttt{RedPT} \arrow[r] & \texttt{PepperT} \arrow[r] & \texttt{VegetableT} \arrow[r] & \texttt{PizzaT}
\end{tikzcd}$$
...

Fig. 6 A part of the list of all merged lists

3.3. RDF/XML-Manchester OWL syntax converter

Since the key to successfully constructed DSL is a proper understanding of the source ontology, OntoP provides a description of all ontology components. Descriptions are written in Manchester OWL syntax.

Visualization of component information is important because if the ontology is better understood, the resulting DSL is also improved. Manchester OWL syntax was chosen for its easy readability, as easily readable syntax shortens the development-cycle.

Manchester OWL syntax was primarily intended for representation and editing of class descriptions. It can also be used for the representation the entire ontology. This means that full descriptions can be prepared for classes, properties, and individuals. A full description of the VegetarianPizza class is presented in Fig. 7.

Class:
VegetarianPizza

Equivalent classes:
Pizza **and** hasTopping **only** (CheeseTopping **or** VegetableTopping)

Disjoint classes:
NonVegetarianPizza

Fig. 7 VegetarianPizza class description written in Manchester OWL syntax

In general OntoP displays those individual ontology components marked with:

- equivalent classes, superclasses, inherited anonymous classes, members, disjoint classes, comments, and labels (classes),
- characteristics, domains, ranges, inverse properties, and super properties (object properties),
- characteristics, domains, ranges, and super properties (datatype properties), and
- types, data property assertion, and object property assertions (individuals).

The conversion between RDF/XML and Manchester OWL syntax is done by the RDF/XML-Manchester OWL syntax converter, which is a part of the OntoP implementation. For the creation of annotation classes (equivalent classes and superclasses), RDF/XML-Manchester OWL syntax converter uses class-description syntax, as represented in Table 1.

Table 1 The class description syntax [21]

OWL Construct	DL	Man. OWL
intersectionOf	$C \cap D$	C **AND** D
unionOf	$C \cup D$	C **OR** D
complementOf	$\neg C$	**NOT** C
oneOf	$\{a\} \cup \{b\} \ldots$	$\{a\ b\ \ldots\}$
someValuesFrom	$\exists R C$	R **SOME** C
allValuesFrom	$\forall R C$	R **ONLY** C
minCardinality	$\geq N R$	R **MIN** N
maxCardinality	$\leq N R$	R **MAX** N
cardinality	$= N R$	R **EXACTLY** N
hasValue	$\exists R \{a\}$	R **VALUE** a

Annotation (equivalent classes) for the VegitarianPizza class, which is shown in Manchester OWL syntax in Fig. 3 is created by the converter within the sequence presented in Fig. 8.

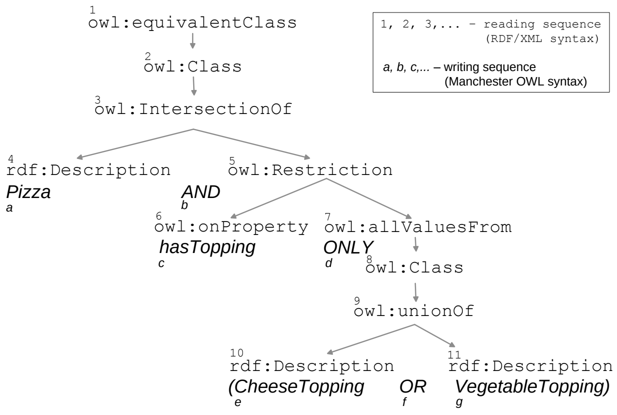
1 owl:equivalentClass
2 owl:Class
3 owl:IntersectionOf
4 rdf:Description Pizza a
5 owl:Restriction AND b
6 owl:onProperty hasTopping c
7 owl:allValuesFrom ONLY d
8 owl:Class
9 owl:unionOf
10 rdf:Description (CheeseTopping e
OR f
11 rdf:Description VegetableTopping) g
1, 2, 3,... – reading sequence (RDF/XML syntax)
a, b, c,... – writing sequence (Manchester OWL syntax)

Fig. 8 The sequence of creating annotation (equivalent classes) for the VegetarianPizza class, which is written in Manchester OWL syntax

3.4. ONTOLOGY METRICS

Several ontologies that can be freely downloaded online could have been used for the presented purposes. When determining whether an ontology was useful, quality became the deciding factor. To make the process easier, new or summarised existing metrics [23], [24] were defined that highlight ontology characteristics. Based on the information on ontology characteristics, the DSL engineer can make a quick and effective decision on the appropriateness of the ontology. For instance, when reviewing information on which components are in the ontology, DSL engineer can determine which types of productions can be generated from the ontology. The metrics used here are considered to be a helpful tool and are not meant to be the final grade by which the ontology is evaluated as appropriate or not. The

Unauthenticated | 194.138.39.60
Download Date | 1/15/14 3:04 AM
VERSITA EMERGING SCIENCE PUBLISHERS

represented metrics are evaluated and presented to users within the OntoP component. This paper continues with the representation of several metrics used in OntoP, explaining their value for DSL development. The metrics are:

- **Number of Classes** (*NoC*)
  Number of Classes is the number of classes within the ontology.

- **Number of Object Properties** (*NoOP*)
  Number of Object Properties is the number of object properties within the ontology.

- **Number of Datatype Properties** (*NoDP*)
  Number of Datatype Properties is the number of datatype properties within the ontology.

- **Number of Indivuduals** (*NoI*)
  Number of Indivuduals is the number of individuals within the ontology.

  All of the above-mentioned metrics are used as maximization fitness evaluations: higher numbers are better for the successful development of DSL. It is also very favorable if the ontology contains all of these components; as all of the metrics are evaluated as positive (higher than zero). It should be noted that high values do not guarantee that the DSL will be successfully developed. On the other hand, low values do not automatically mean that the ontology is insufficient for the DSL development.

- **Number of Root Classes** (*NoRC*)
  Number of Root Classes [24] is the number of classes explicitly defined within the ontology. Root-class is a class that does not have any superclass. Mathematically *NoRC* can be formulated as:

$$NoRC = \sum_{i=1}^{n} C_i, \qquad C_i = \begin{cases} 1 & \text{if } C_i \text{ is root class} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

  where: C - the ontology class, n - number of classes within the ontology. The number of classes can be from 1 to n, because the ontology can contain only root-classes.

- **Number of Leaf Classes** (*NoLC*)
  Number of Leaf Classes [24] is the number of leaf-classes explicitly defined within the ontology. A leaf-class is a class that does not have any subclasses. Mathematically *NoLC* can be formulated as:

$$NoLC(O) = \sum_{i=1}^{n} C_i, \qquad C_i = \begin{cases} 1 & \text{if } C_i \text{ is leaf class} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

  where: C - the ontology class, n - number of classes within the ontology. The number of leaf classes can be from 1 to n. A root-class that does not have any subclasses can be considered as leaf-class.

  With both metrics, *NoRC* and *NoLC*, border values are undesirable for DSL development. For instance, values close to 0 in both would mean that the ontology only has classes on the first level of the class hierarchy. Consequently the Ontology2DSL framework can only generate the start production. Automated generation of productions based on subclasses is impossible in such a case [12], [13].

- **Average Depth of Inheritance Tree of Leaf Nodes** (*ADIT-LN*)
  Average Depth of Inheritance Tree of Leaf Nodes [24] is the relation between the sum of depths of all paths and the total number of all paths. The depth is the number of all nodes between the root and leaf-nodes along a path. All ontology-paths are all the different paths from each root-node to individual leaf-nodes. The root-node is the first level on any path. Mathematically *ADIT-LN* can be formulated as:

$$ADIT\text{-}LN = \sum_{i=1}^{n} D_i/n \tag{3}$$

  where: D_i - depth of $i-th$ path, n - number of all possible paths.

- **Average Population** (*AP*)
  Average population [23] is the ratio between the number of individuals and number of classes. Mathematically *AP* can be formulated as:

$$AP = i/c \tag{4}$$

  where: i - number of all individuals within ontology, c - number of all classes within ontology.

- **Class Richness** (*CR*)
  Class Richness [23] is the ratio between the number of classes with individuals and the number of all classes within the ontology. Mathematically CR can be formulated as:

$$AP = c'/c \tag{5}$$

  where: c' - number of classes with individuals, c - number of all classes within ontology. The higher the value of *AP* the more knowledge the ontology contains.

  For DSL, the development values of *AP* in *CR* should be high since this provides a better chance of developing a richer grammar.

- **Relationship Richness** (*RR*)
  Relationship Richness [23] is the ratio between the number of relations within the ontology on one side, and the sum of all subclasses (number of inheritance relationships) added by number of relations on the other side. Mathematically *RR* can be formulated as:

$$RR = r/(sc+r) \tag{6}$$

Unauthenticated | 194.138.39.60
Download Date | 1/15/14 3:04 AM
VERSITA EMERGING SCIENCE PUBLISHERS

where: r - number of all relationships, sc - number of all subclasses (inheritance relationships) within ontology. This metric represents the diversity and the position of relationships within the ontology. Ontology that contains more relationships that are not of class-subclass type (RR value is close to 1) is richer than a taxonomy (which contains only class-subclass relationships and has RR value close to 0).

Higher values of $ADIT$-NL, AP, CR in RR are desirable for DSL development. Since higher values indicate information rich ontologies they can be used to create richer, better grammar and programs.

- **Inheritance richness (*IR*)**
  Inheritance richness [23] is defined as the average number of subclasses per class. Mathematically IR can be formulated as:

$$IR = sc/c \tag{7}$$

where: sc - number of all subclasses within ontology, c - number of all classes within ontology. This metric measures the distribution of information on different levels of the class hierarchy, and serves as an indicator of how well the knowledge is grouped into different categories and sub-categories within the ontology. IR metric distinguishes between the horizontal and vertical ontology. Horizontal-ontology has classes that have many subclasses, whilst vertical-ontology has classes that have a small number of direct subclasses. IR values close to 0 indicates vertical ontologies (describing a specific domain in detail) whilst high IR values represent horizontal-ontologies (more general knowledge with fewer details).

Lower values are desired for DSL development because they represent vertical ontologies that describe a specific domain in detail.

4. CONCLUSION AND FUTURE WORK

When working with OWL ontologies, a large set of tools exists that enable the creation, editing, reasoning over ontologies, and the usages of ontologies for various applications. This paper presented another parser for working with ontologies: the OntoP ontology parser. OntoP was developed for the needs of a larger framework: Ontology2DSL. OntoP, within the framework Ontology2DSL, is tasked with four tasks: filling of the data structure, construction and representation of hierarchies, retrieval and displaying of information on individual ontology components, and enabling the evaluation of ontologies with metrics.

The accuracy of OntoP has been verified on a set of ontologies. Information retrieved by OntoP has been compared to information extracted from Protégé. OntoP has been proven to be equal to Protégé. Experiment that will show the practical value of metrics will be part of our future work. Future work will include a separation of the OntoP parser from the Ontology2DSL framework, so that it can be offered as a freely-accessible tool for interested researchers.

ACKNOWLEDGEMENT

This work is sponsored by bilateral project ”Language Patterns in Domain-specific Languages Evolution” between Slovenia (Grant No. BI-SK/11-12-011) and Slovakia (Grant No. SK-SI-0003-10).

REFERENCES

[1] LACY, L.: OWL: Representing Information Using the Web Ontology Language. Trafford Publishing, 2005, ISBN 1412034485.

[2] HEBELER, J. – FISHER, M. – BLACE, R. – PEREZ-LOPEZ, A.: Semantic Web Programming. Wiley Publishing, 2009, ISBN 047041801X.

[3] HORRIDGE, M.: A Practical Guide to Building OWL Ontologies Using Protégé 4 and CO-ODE Tools. http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf.

[4] STUDER, R. – BENJAMINS, R. V. – FENSEL, D.: *Knowledge engineering: Principles and methods*, Data & Knowledge engineering **25**, No. 1-2 (1998) 161–197. http://www.sciencedirect.com/science/article/pii/S0169023X97000566

[5] STAAB, S. – STUDER, R.: Handbook on Ontologies. Springer Verlag, 2009, ISBN 3540408347

[6] Protégé, http://protege.stanford.edu/

[7] KALYAMPUR, A. – PARSIA, B. – HENDLER, J.: *A tool for working with web Ontologies*, International Journal on Semantic Web and Information Systems **1**, No. 1 (2005) 36–49. http://www.ijswis.org/?q=node/8

[8] SIRIN, E. – PARSIA, B. – GRAU, B. – KALYANPUR, A. – KATZ, Y.: *Pellet: A practical OWL-DL reasoner*, Web Semantics: Science, Services and Agents on the World Wide Web **5**, No. 1 (2007) 51–53.

[9] TSARKOV, D. – HORROCKS, I.: FaCT++ Description Logic Reasoner: System Description. In Proceedings of the International Joint Conference on Automated Reasoning, Seattle, pp. 292–297, Aug. 2006.

[10] BECHHOFER, S. – VOLZ, R.– LORD, P.: Cooking the Semantic Web with the OWL API. In Proceeding of the First International Semantic Web Conference, pp. 659–675, Oct. 2003.

[11] Jena, http://jena.sourceforge.net/

[12] ČEH, I. – ČREPINSEK, M. – KOSAR, T. – MERNIK, M.: *Ontology Driven Development of Domain-Specific Languages*, Computer Science and Information Systems **8**, No. 2 (2011) 317–342. http://www.comsis.org/archive.php?show=vol0802

[13] ČEH, I. – ČREPINŠEK, M. – KOSAR, T. – MERNIK, M. – HENRIQUES, P. R. – PEREIRA, M. J. V. – DA CRUZ, D. – OLIVEIRA, N.: Tool-supported building of DSLs from OWL ontologies. INForum 2011 : terceiro simpósio de informática, Coimbra, pp. 210–221, Sep. 2011.

Unauthenticated | 194.138.39.60
VERSITA EMERGING SCIENCE PUBLISHERS
Download Date | 1/15/14 3:04 AM

[14] AHO, A. V. – LAM, M. S. – SETHI, R. – ULLMAN, J. D.: Compilers: Principles, Techniques, and Tools. Addison Wesley, 2007, ISBN 0201100886.

[15] MERNIK, M. – HEERING, J. – SLOANE, A. M.: *When and how to develop domain-specific languages*, ACM Computing Surveys **37**, No. 4 (2005) 316–344. http://dl.acm.org/citation.cfm?id=1118890 &picked=prox

[16] CARDOSO, J.: *The Semantic Web Vision: Where are We?*, Intelligent Systems **22**, No. 5 (2007) 84–88, http://ieeexplore.ieee.org/xpl/tocresult.jsp? isnumber=4338482

[17] ANTONIOU, G. – VAN HARMELEN, F.: A Semantic Web Primer, second edition. The MIT Press, 2008, ISBN 0262012421.

[18] BAADER, F. – CALVANESE, D. – MCGUINNESS, D. – NARDI, D. – PATEL-SCNEIDER, P. F.: The description logic handbook: Theory, implementation and applications. Cambridge University Press, 2003, ISBN 0521781760.

[19] BECHHOFER, S. – VAN HARMELEN, F. – HENDLER, J. – HORROCKS, I. – MCGUINESS, D. L. – PATEL-SCHNEIDER, P. F. – STEIN, L. A.: Owl Web Language Reference. http://www.w3.org/TR/owl-ref/

[20] PATEL-SCHNEIDER, P. F. – HAYES, P. – HORROCKS, I.: OWL Web ontology Language Semantics and Abstract Syntax. http://www.w3.org/TR/owl-semantics/

[21] HORRIDGE, M. – DRUMMOND, N. – HOODWIN, J. – RECTOR, A. – STEVENS, R. – WANG, H. H.: The Manchester OWL syntax. OWL: Experiences and Directions Workshop, Athens, Georgia, USA, Nov. 2006.

[22] SMITH, M. K. – WELTY, C. – MCGUINNESS, D. L.: OWL Web ontology Language Guide. http://www.w3.org/TR/owl-guide/

[23] TARTIR, S. – ARPINAR, I. B. – SHETH, A. P: Ontological evaluation and validation. http://knoesis.wright.edu/library/

[24] YAO, H. – ORME, A. M. – Etzkorn, L.: *Cohesion Metrics for Ontology Design and Application*, Journal of Computer Science **1**, No. 1 (2005) 107–113. http://thescipub.com/issue-jcs/1/1

Received January 30, 2012, accepted March 30, 2012

BIOGRAPHIES

Ines Čeh received her B.Sc. degree in computer science at the University of Maribor, Slovenia in 2008. Her research interests include domain-specific languages and ontologies. She is currently a Ph.D student, employed as a researcher at the University of Maribor, Faculty of Electrical Engineering and Computer Science.

Milan Zorman received his Ph.D. in computer science in 2001 at the University of Maribor. At the moment he works as an Associate Professor at the Faculty of Electrical Engineering and Computer Science, the Medical Faculty and Faculty of Health Sciences, University of Maribor. He is also the director of Centre for Interdisciplinary and Multidisciplinary Research and studies of the University of Maribor (CIMRS), and head of the Enterprise Europe Network project at the same institution. As a researcher he is active in the fields of artificial intelligence, machine learning, medical and nursing informatics, data mining, knowledge extraction, hybrid intelligent systems and complex systems.

Matej Črepinšek received his Ph.D. degree in computer science at the University of Maribor, Slovenia in 2007. His research interests include grammatical inference, evolutionary computations, object-oriented programming, compilers, grammar-based systems and Android application development. He is currently a teaching assistant at the University of Maribor, Faculty of Electrical Engineering and Computer Science.

Tomaž Kosar received his Ph.D. degree in computer science at the University of Maribor, Slovenia in 2007. His research is mainly concerned with design and implementation of domain-specific languages. Other research interest within computer science include also domain-specific modelling languages, empirical software engineering, software security, generative programming, compiler construction, object-oriented programming, object-oriented design, refactoring, and unit testing. He is currently a teaching assistant at the University of Maribor, Faculty of Electrical Engineering and Computer Science.

Marjan Mernik received his M.Sc. and Ph.D. in computer science from the University of Maribor in 1994 and 1998, respectively. He is currently Professor of Computer Science at the University of Maribor. He is also Visiting Professor of Computer and Information Sciences at the University of Alabama, Birmingham USA, and at the University of Novi Sad, Faculty of Technical Sciences. His research interests include programming languages, compilers, domain-specific (modeling) languages, grammar-based systems, grammatical inference, and evolutionary computations. He is a member of the IEEE, ACM and EAPLS.

Jaroslav Porubän is an associate Professor at Department of Computers and Informatics, Technical university of Košice, Slovakia. He received his MSc. in Computer Science in 2000 and his Ph.D. in Computer Science in 2004. Since 2003 he is the member of the Department of Computers and Informatics at Technical University of Košice. He was involved in the research of profiling tools for process functional programming language. Currently the main subject of his research is the computer language engineering, concentrating on the design and implementation of domain-specific languages and computer language composition and evolution.

Unauthenticated | 194.138.39.60
Download Date | 1/15/14 3:04 AM
VERSITA EMERGING SCIENCE PUBLISHERS