OWL for the Masses: From Structured OWL to Unstructured Technically-Neutral Natural Language

Ljubinka Gareva - Takasmanov
Department of Computer Science
CITY College
Thessaloniki, Greece
e-mail: lgareva@city.academic.gr

Ilias Sakellariou
Department of Applied Informatics
University of Macedonia of Econimic and Social Sciences
Thessaloniki, Greece
e-mail: iliass@uom.gr

***Abstract*—The Web Ontology Language (OWL) is one of the fundamental blocks for realizing the Semantic Web vision, since it allows to represent probably the most difficult component of the latter; knowledge regarding the concepts of domain of discourse, i.e. ontologies. Unfortunately, its expressive power goes in hand with a rather verbose syntax, difficult to be understood by non-technical users, and thus leading to difficulties in validation and verification of the represented knowledge. A translation tool from OWL to some form of natural language could significantly assist users towards such tasks. Such a tool requires correctly interpreting the ontology constructs, representing highly nested ontologies, and forming logical sentences. This paper presents an innovative approach to directly translating RDF/XML based structured ontology into error free concise natural language text akin to English.**

I. INTRODUCTION

The inability of current Web services to manage and deliver machine processable information, results in Internet information overloading. The Semantic Web is a vision according to which web content is given meaning so that is easily processable by machines. By "understanding" and integrating the knowledge, machines would finally be capable of presenting concise and valuable information to web users. The Web Ontology Language (OWL) is one of the fundamental building blocks of the future web, allowing explicit representation of terms and their relationships in the modeled domain, i.e. their semantic annotation. Unfortunately, the OWL language is known for its verbose syntax, a feature that poses difficulties to Semantic Web users (and even domain experts) in understanding and validating encoded knowledge. One promising solution to this obstacle is to provide translation tools from OWL representation to a more readable, user-friendly form.

However, existing tools provide translations that are either apparently incomplete or contain a number of grammatical and logical irregularities. Furthermore, nesting of ontologies is considered as difficult to solve issue that further contributes towards the inconsistency of translation. Our motivation was to find a way to overcome both problems and enable error free and concise translation of a given ontology-based document written in OWL into a concise form that resembles natural language text. Towards this, the approach we have followed is essentially a two stage process: parsing a given ontology and generating an intermediate model of the OWL constructs and applying predefined translation rules on that model to obtain the desired translation. Our goal is to provide a tool that would aid in the advancement of the future web, by allowing more non-technical users to comprehend, verify and validate OWL encoded ontologies.

The rest of the paper is organized as follows. In section 2, a brief introduction to OWL language is given. Section 3 shows the architecture modules, while Section 4 reveals some implementation details. Section 5 contains the related work and a discussion, whereas section 6 presents some findings from the tool testing. Finally, in section 7, the conclusion and future work are presented.

II. RECOGNIZING WEB ONTOLOGY LANGUAGE

The aim of the Semantic Web initiative is to construct a global medium for interconnected data exchange based on sharing, processing, integrating, and reusing volumes of data, not documents, by agents, as stated by Tim Berners Lee [1]. Semantic Web has a goal not to interpret the meaning of documents written in English language, rather to mark up the existing HTML documents, thus give them meaning, in order to allow the machines to "understand" their content [2].

OWL is a language for defining and instantiating ontologies [3]. Recommended by W3C (World Wide Web Consortium), OWL formal semantics include specifications for deriving logical consequences of an ontology, which are facts and knowledge not explicitly stated but inferred by the semantics [3, 4]. Being a part of the Semantic Web vision, OWL has to enable resources describing Web content to be added on the Web and to be obtained from different sources, by relating ontologies and importing different ontology information [3]. Based on its open world assumption, OWL allows anybody to additionally define resources by

providing ontology-related information.

OWL is syntactically comprised of entities, expressions, and axioms. Entities are all individuals interpreted through classes, and all properties that are inherited by individuals as members of a particular class [5]. Expressions present a collection of terms (attributes) and their inter-relations which model specific domain of the world, thus describing the entities [5, 6]. Axioms assert the validity of entities in the domain [5].

There exist three versions of OWL: *OWL Full*, an extension of RDF that includes the *full* set of OWL language primitives and allows full primitives manipulation; *OWL DL* (Description Logics), that imposes constraints on OWL Full limiting complexity and ensuring that the language is based on well defined description logics; and *OWL Lite*, which further constrains DL thus increasing the easiness of implementation and understanding, but at the same time limiting expressiveness [3].

III. OWL FOR THE MASSES ARCHITECTURE

The approach followed in order to translate the complex OWL syntax to the desired form, requires:

- definition of general translation rules, applicable to any ontology,
- parsing of the ontology in order to identify the building constructs and their relationships. All superfluous text such as comments, labels, annotations, and parts of namespaces must be ignored.
- creation of an intermediate representational model based on the parsed constructs
- application of the predefined rules to the representational model to obtain the translational model in the form of sentences written in natural language English text.

Thus, the architecture followed consists of two main layers or modules as referred to henceforth (Figure 1):

- The *Builder module*, which parses the given ontology and subsequently builds the representational model, mentioned above.
- The *Lexer module*, which takes as input the intermediate model and by applying the translation rules, generates the ontology translation and presents the output.

A. The Builder Module

The **Builder** module is responsible for parsing the loaded ontology written in RDF/XML format and creating the representational model that contains all OWL constructs in the ontology. The current implementation considers as main ontology constructs the following: *Classes*, *Individuals*, *Object Properties*, and *Datatype Properties*. Identified relations between the main constructs can be any restrictions on properties, set operations, property characteristics, relations among classes, as well as enumerated classes, as those are defined by the OWL language. Its operation is described in the paragraphs that follow.

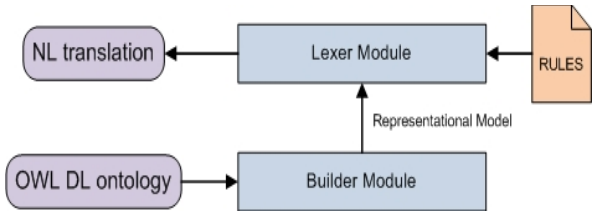

Figure 1. Illustration of the Architecture Modules

The parsing module consists of one main class **Builder** and a set of helper classes. While the **Builder** class is responsible for discovering the main OWL constructs, the other classes are accountable for detecting the relationships among OWL constructs in the nested RDF/XML hierarchy and creating the overall representational model.

Parsing is invoked by the class **Builder** which preserves all encountered OWL constructs as classes. This class contains methods for managing every OWL construct. When a new OWL construct is encountered during the parsing, a new construct class of the corresponding type is created and stored in a list. Upon creation, the task of parsing the ontology and expanding the representational model is passed to the newly created class instance. The latter is capable of resolving the relations among itself and the related OWL constructs such as OWL Classes, Object Properties, and Datatype Properties.

In a similar manner, each of the OWL construct classes defines methods for managing relations with other constructs as classes and properties. For example, the *GenericClass*, instance, defines methods for handling restrictions on properties, equivalent classes, disjoint classes and subclasses. As in the case of class **Builder**, restrictions and equivalent classes are created and preserved as class instances of the corresponding *Restriction* and *EquivalentClass,* passing to them the responsibility for further parsing and supplementing the representation model. The *Restriction* class, on the other hand, defines methods for all relations found through parsing the nested RDF/XML structure as child nodes of the *owl:Restriction* tag.

The process of generating the representational model continues in similar manner, with each class instance contributing in parsing and building parts of this model. Table I, depicts part of the generated the representational model, or otherwise referred as parse tree of the Pizza ontology.

All aforementioned classes inherit the invokeContext method from the AbstractClass. This method is crucial since it is responsible for invoking a specific handler method defined within a class depending on the context (relation) encountered during parsing. For example, for the GenericClass class, this method invokes the owl_Restriction handler method for restrictions, the owl_disjointWith on encountered disjoint classes, etc.

TABLE I. PARTIAL REPRESENTATIONAL MODEL

```
Parse tree
•QuattroFormaggi
  -disjoints Siciliana , UnclosedPizza , Rosa
  -parents NamedPizza
  -name QuattroFormaggi
    •restriction
   -type onproperty
   -value hasTopping
    •abstract(AllValuesFrom)
     •abstract(UnionOf)
       •FourCheesesTopping
        -disjoints ParmesanTopping,
MozzarellaTopping , GoatsCheeseTopping
        -parents CheeseTopping
        -name FourCheesesTopping
             •restriction
          -type onproperty
          -value hasSpiciness
           •abstract(SomeValuesFrom)
            •Mild
             -parents Spiciness
             -name Mild
      •TomatoTopping
        -parents VegetableTopping
        -name TomatoTopping
```

This modular approach allows easy expansion of the module for managing relations among OWL constructs through definition of the corresponding classes for any newly added constructs, as well as internal methods for parsing and preserving the related inner relations as parameters.

B. The Lexer Module

The **Lexer** module is responsible to provide a natural language representation of the ontology. The application does not employ any Part-Of-Speech (POS) tagger to classify the words extracted from the ontology. Instead, the Lexer generates a set of sentences or natural language expressions that correspond to the ontology constructs and relations by applying a set of translation rules on the representational model generated, in the previous step.

Translation rules are designed to be applicable to different ontology subsets of OWL constructs and associated relations that identify specific logical expressions. An example of such expression would be an OWL class disjoint with a set of other classes defined in the ontology. In addition, most of these logical expressions require a set of rules to be applied to achieve correct and more readable translation.

The rules' syntax consists of two parts separated by the character "|". The left part represents either an OWL construct class, such as *GenericClass*, *Individual*, *ObjectProperty*, or *DataTypeProperty*, or a term defining some parameters found in the OWL construct class attributes, such as disjoint classes, parent class or object instances of the classes representing the relations with other OWL constructs. The right part is a set of constant strings, template variables denoted by "$", and expressions enclosed in "< >" that contain parameters (attributes) as defined in the representational model. These parameters may either be a single element, or a list, in which case are enclosed with curly brackets. Additionally, the syntax allows utilizing Python built-in expressions and functions as parameters denoted with "@" to enable term manipulation and the required NL translation.

The Lexer module parses these rules, ignoring any comments starting with "#", separates the left and right parts as rule name and rule value, respectively, and stores them in a rules repository. Given the representational model, this module iterates through the class instances that correspond to the OWL constructs and applies the rules where the rule name matches the current class instance type.

For example, the following rule is defined for *GenericClass* instances.

GenericClass | I hereby declare $className.
<ParentClass{parameters["parents"]}>
<Disjointed{parameters["disjoints"]}> <Objects{objects}>

During the translation process, the template variable **$className** is replaced by the name of the OWL construct instance being processed at that time. It should be noticed that the expressions *ParentClass{parameters["parents"]}* and *Disjointed{parameters["disjoints"]}* are handled by corresponding rules. The *Objects{objects}* expression refers to the class instances from the representational model defining the relations with other constructs, such as *Restrictions*, *AllValuesFrom*, *SomeValuesFrom*, *OneOf* etc. For each of these instances separate rules are being defined.

In the *ParentClass* expression, template variables **$sender** and **$className** are substituted with the *sender* construct instance invoking this rule and the name of its parent (superclass) respectively:

ParentClass | $sender is $className.

In specific cases, OWL construct indexing is specified within the rule groups, to facilitate NL expressions comprised of several OWL constructs of same type, such as disjoint OWL classes. Indexing allows easier rule manipulation and straightforward generation of the desired NL output for the specific subset. The disjoint OWL classes rule example presented below depicts indexing, with S as start index, F as final index and * for the rest of the constructs in-between S and F.

Disjointed:S | $sender is not $className
Disjointed: | , $className*
Disjointed:F | nor $className.

Table II presents examples of natural language sentences generated by the application of the rules described above.

TABLE II. GENERATED NL OUTPUT BASED ON TRANSLATION RULES

Generated NL output	The rule applied
I hereby declare Fish Topping.	GenericClass | I hereby declare $className. <ParentClass{parameters["parents"]}> <Disjointed{parameters["disjoints"]}>
Fish Topping is Pizza Topping.	ParentClass | $sender is $className.
Fish Topping is not Sauce Topping, Herb Spice Topping, Fruit Topping, Nut Topping, Vegetable Topping nor Meat Topping.	Disjointed:S | $sender is not $className Disjointed:* | , $className Disjointed:F | nor $className.

The syntax defined endorses usage of Python built-in functions as parameters , annotated by "@". In the example that follows, the **$ownerclass** template variable is replaced with the parent class, which is always a Restriction class instance. The part of the expression *parent.parameters["value"]* returns the property on which this restriction applies (e.g. hasTopping), whereas the replace function (built-in Python function), replaces "has" with "has at least one", thus generates the desired NL output.

SomeValuesFrom | $ownerclass <@parent.parameters["value"].replace("has","has at least one")+" that is"> <Class{objects}>.

The final NL translation resulting from the application of the above rule for *SloppyGiuseppe* class instance would be:

Sloppy Giuseppe has at least one topping that is Tomato Topping.

IV. IMPLEMENTATION

For the implementation the Python programming language was employed. Besides Python's scripting language features, that allow simple text and file processing, an essential reason for using this language was the support of DOM API [11]. DOM is endorsed by W3C as a cross reference platform for manipulating XML documents. More specifically, the application uses the lightweight implementation of DOM, *minidom*, which is smaller and simpler to apply. The tool's implementation relies on the provided XML parser to generate the parse tree. Finally, *OWL for the Masses* uses the Python Regular Expressions module for rule application.

At the moment, the tool can be used either through a CLI (Command Line Interface) on any OS platform with Python interpreter installed, or as a server-side application running on a web server. Figure 2 presents an example of the famous Pizza ontology translated by the OWL for the Masses tool.

V. EVALUATION AND RELATED WORK

Existing tools offer a variety of services that aid computer scientists to Semantic Web development tasks. However, their interfaces, output, and usage approach requires a steep learning curve for non-technical Web users.

Ginseng [7] is a natural language search engine that enables concise ontologies querying using plain English. GINO [8], an extension of Ginseng, is an ontology editor that supports querying and editing access to any ontology oriented knowledge repository using a language that resembles English. Both of these tools are restricted to the vocabularies defined by ontologies already stored in Jena inference model, thus limited when querying other ontologies [7, 8].

Swoop [9] is Java-based OWL ontology editor and browser that display parts of ontology written in natural English language. However, the code implementing natural language processing is not fully completed and thus, the translation provided is ambiguous, inconsistent, and not as natural as the English language. The translated strings contain multiple repetitions of unwanted words and experience minor grammatical problems. More importantly, the tool does offer NL phrasing for individuals and properties, and Disjoints of classes, along with complex sentence formation are not supported. Additionally, Swoop has problems with paraphrasing nested ontologies.

OWL Verbalizer translates RDF/XML ontology into a subset of English, called Attempto Controlled English (ACE). For complex class description representations relative clauses are nested in other relative clauses; nevertheless, embedding is possible to only few nesting levels. The application version reviewed at the time of writing does not support data valued properties and complex class description. Furthermore, only logical parts of ontology are translated to ACE, not taking in consideration many constructs, annotations, imports, versioning, and other important OWL content [10]. Additional ambiguity is noted in the URIs' names, as they are truncated and only the last word is translated. What is more, equivalent properties are usually perceived as equivalent classes. Verb tenses are not correctly displayed, and some grammatical inconsistencies are encountered.

In our opinion, OWL for the Masses introduces an original and innovative approach towards translating OWL into concise English. The tool independently parses ontologies and builds the representational model from the OWL constructs without relying to existing parsers and libraries, thus there are no dependencies and limitation posed by third party tools. Its modular implementation

allows easily adding constructs by extending the rule set with the corresponding definitions, thus allowing to effortlessly manage any missing relations and solving the problem of ambiguous and incomplete translations. This architecture, contributes towards overcoming many difficult issues such as grammatical inconsistencies, multiple repetitions of unwanted words, and complex sentence formation.

Since POS taggers and complex grammatical rules are not employed, any changes/corrections of the rules' syntax is rather straightforward, which can be considered as an advantage compared to other tools. Finally, by representing the ontology in an intermediate form as a parsed tree, OWL for the Masses has the ability to overcome the issue of translating highly nested ontologies.

Currently, the tool is able to load and translate all correctly written OWL DL ontologies. Ontologies lacking proper header and namespaces are not translated and parsed. Obviously, OWL Lite ontologies are also translated by the application since every OWL Lite ontology is an OWL DL ontology.

VI. Testing Results

In order to assess the completeness of the tool and the consistency of the translation, the application was introduced to two groups of potential users: a) professionals from different fields; b) stakeholders, and other moderately computer literal people.

In the first group, five professionals familiar with OWL syntax and semantics, were expected to verify the translation correctness by comparing the structure of the ontology with the translated natural language sentences, to assess the grammatical accuracy and the logical consistency of the translation. Furthermore, they were asked and to recognize the structure of the ontology constructs, evaluating whether these constructs are correctly written in RDF/XML syntax and whether some logical inconsistencies identified in the input might produce errorneous output.

The second group, consisting of ten stakeholders and fairly computer literal people not familiar with OWL, was encouraged to read the output of the application and to comment on the translation. The evaluators were asked to assess the accuracy and reliability of the translation, and to report on the usability of the tool, the execution time, translation correctness, as well as whether the translation was grammatically and logically right.

Both groups of evaluators evaluated two OWL DL ontologies and reported the following: the translation was generated fast, in three to five seconds depending on the ontology size and machine configuration. The sentence formation was assessed by professionals and ordinary users as logical, grammatically correct, and syntactically coherent. The evaluators belonging to the second group concluded that the translation context was rational and concise.

The first group of users was generally very satisfied from the translation when compared to the RDF/XML written constructs. Nevertheless, almost all evaluators pointed out that comments, labels, and annotations are left as unparsable nodes, which is since it was assumed that these nodes do not contribute towards logical meaning of the ontology and are useless in the translation process. Moreover, more than half of the users from this group found missing translations for very few of the constructs. This is due to the fact that not all OWL DL constructs were implemented, but only those most frequently used. It is interesting to note that almost every professional user observed repeated sentences with similar meaning in the translation, which was perceived as correct since the loaded ontology contained repeating constructs. The list with unparsed nodes displayed along with translation was considered as quite useful when evaluating the translation completeness. The tool, in general, was evaluated as usable, easy to operate with, and with user-friendly interface.

VII. Conclusion and Future Work

By developing an application for translating OWL ontologies, the community is offered a tool for naturally presenting highly organized machine processable knowledge to humans. We consider the latter important, since the wide adoption of the Semantic Web would soon necessitate non expert users to encode, review and verify knowledge available through ontologies. The tool is able to present a logical and correct translation of constructs, manage complex OWL expressions and directly translate ontologies in human-understandable language.

One of the main issues that we consider addressing in the near future refers to extending the set of translated constructs with all OWL DL constructs that were not initially implemented. The modular structure of the application described above, allows effortless expansion by simply adding new constructs and relations, followed by appropriate translation rules.

A further enhancement concerns parsing and translation of user-defined constructs. This feature along with the tool's ability to recognize alternative syntactic forms would make the application supportable for ontologies written in subsets of OWL Full.

Supporting OWL 2 constructs, is another of our main targets that would possibly allow translation of any ontology found on the Web as well as achieving completeness on the set of translation rules.

As far as further usability is concerned, the application could include a module for testing the correctness of OWL ontologies, thus presenting to the user any possible errors and inconsistencies residing in the ontology under investigation. As a result, novice users would be able to

easily learn and understand OWL intricate syntax. With slight modifications, the tool could be utilized as a mean to educate people in the OWL language. As a consequence, there will be an increased community of users with gained trust and acceptance of newest Semantic Web standards, which could lead to a faster adoption and expansion of the future web.

REFERENCES

[1] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities", *Scientific American, May 2001*, pp. 34–43.

[2] T. Berners-Lee, "Semantic Web Roadmap", http://www.w3.org/DesignIssues/Semantic.html.

[3] W3CRecommendation, "OWL Web Ontology Language Overview", http://www.w3.org/TR/owl-features/.

[4] W3CRecommendation, "OWL 2 Web Ontology Language: Direct Semantics", http://www.w3.org/TR/owl2-semantics/.

[5] W3CRecommendation, "OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax", http://www.w3.org/TR/owl2-syntax/.

[6] W3CRecommendation, "OWL 2 Web Ontology Language: Primer", http://www.w3.org/TR/owl2-primer/.

[7] A. Bernstein, E. Kaufmann, C. Kaiser, " Querying the Semantic Web with Ginseng: A Guided Input Natural Language Search Engine" in The Fifteenth Annual Workshop on Information Technologies and Systems (WITS'05), Las Vegas, 2005.

[8] A. Bernstein, and E. Kaufmann, "GINO - A Guided Input Natural Language Ontology Editor," in The 5th International Semantic Web Conference, Athens, 2006, pp. 144-157.

[9] D. Hewlett, A. Kalyanpur, V. Kolovski, C. Halaschek-Wiener, "Effective NL Paraphrasing of Ontologies on the Semantic Web," in Proceedings of the ISWC 2005 Workshop on End User Semantic Web Interaction Galaway, Ireland, 2005.

[10] K. Kaljurand and N. E. Fuchs, "Verbalizing owl in attempto controlled english," in Proceedings of Third International Workshop on OWL: Experiences and Directions, Innsbruck, Austria (6th-7th June 2007), vol. 258, 2007. [Online]. Available: http://rewerse.net/publications/download/REWERSE-RP-2007-024.pdf

[11] D. Mertz, *Text Processing in Python*, Addison Wesley, 2003.

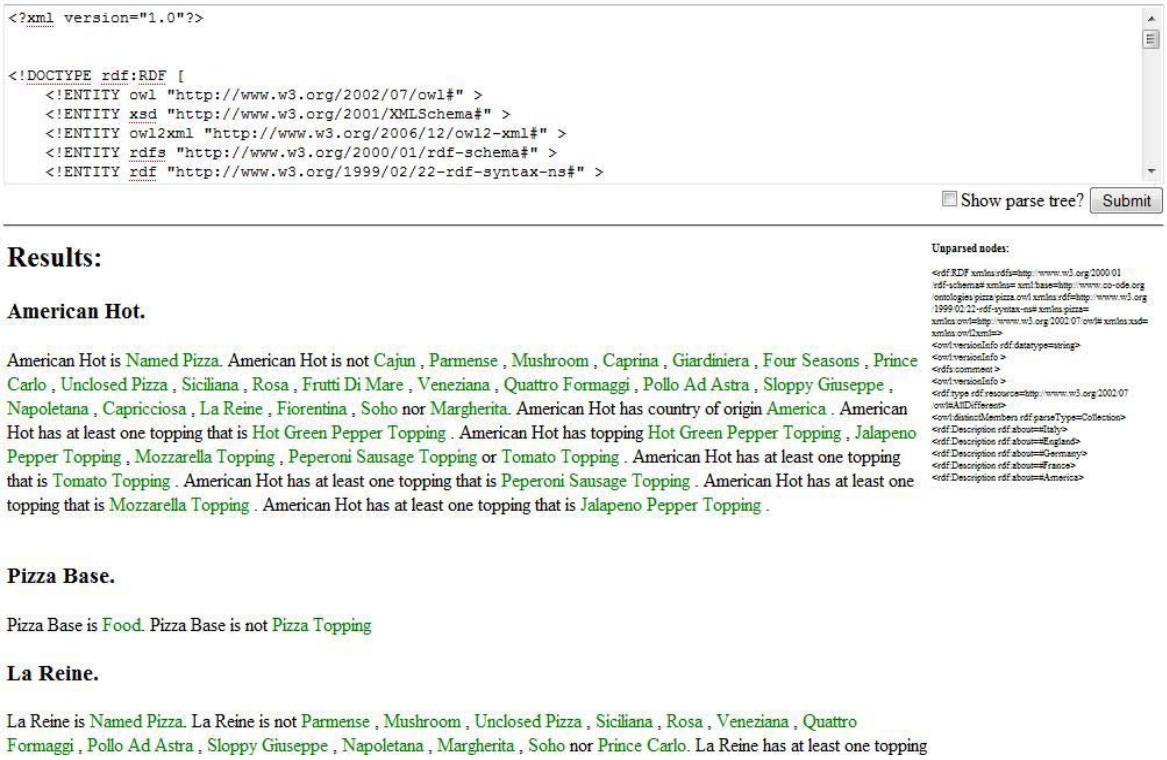

Figure 2. OWL for the Masses: Translating (part) of the famous Pizza Ontology, using OWL for the masses.