RESTful Encapsulation of OWL API

Ramya Dirsumilli[1] and Till Mossakowski[2]
[1]Department of Computer Science, Otto von Guericke University, Magdeburg, Germany
[2]Institute of Knowledge and Language Engineering, Otto von Guericke University, Magdeburg, Germany

Keywords: Web Ontology Language (OWL), OWL API (Application Programming Interface), REpresentational State Transfer, Modularity, Logical Consequences, Consistent Ontology, Ontohub.

Abstract: OWL API is a high level API for working with ontologies. Despite of its functionalities and numerous advantages, it is restricted to a set of users due to its platform dependency. Being built as a java API the OWL API can only be used by java or related platform users. The major goal of this paper is to design a RESTful web interface of OWL API methods, such that ontology developers and researchers independent of platform could work with OWL API. This RESTful OWL API tool is designed to exhibit all the functionalities of OWL API that do not deal with rendering the input ontology such that it doesn't behave as an ontology editor, instead supports web ontology developers and open ontology repositories such as Ontohub.

1 INTRODUCTION

OWL API, a high level java based Application Programming Interface (API) supports creation, manipulation and serialization of OWL ontologies. Being available since 2003, OWL API has undergone a number of changes and now is based on OWL2. Application developers and researchers from many fields have successfully used and are using OWL API to support and work with ontologies. Regardless of its prominence amongst ontology developers, the OWL API still remains domain specific as it is a java API. Java API is a standard interface, which can only be accessed by developers in building java powered applications. This constraint limits the usage of OWL API to a particular technology. But managing and working with ontologies cannot be restricted to a single platform users, instead making it available as a platform independent API could benefit several ontology based applications and web-based ontology repositories like Ontohub, Bioportal, etc. This paper is an attempt towards generating OWL API methods in the form of a web service, such that OWL API can be accessed independent of domain to parse and verify ontologies.

"A web service is a software system designed to support interoperable machine-to-machine interaction over a network" (Booth et al., 2004). It provides a standard mode of inter-operations amongst various software applications that run on different platforms and in different frameworks. Web services are mostly APIs that are built to either extend existing applications, or some times to create a plug-in for an entirely new application. While there are many web services such as SOAP, WSDL, etc., and mechanisms like RPC (Remote Procedure Calls) available to design web-based API, REST (Representational State Transfer) is a lightweight alternative. REST is a web architectural style that is framed with a group of constraints for designing network applications. REST is more likely to be called light-weight due to its platform independency, language independency, running on HTTP standards, and usage in presence of firewalls.

Therefore, using the concepts of RESTful web services and wrapping selected methods of OWL API, the RESTful OWL API is designed. Selected methods that do not deal with rendering the ontology are encapsulated to exhibit functionalities. And the encapsulated methods are encased into web service such that they exhibit RESTful features. The RESTful OWL API comprises of 14 different methods that are designed on top of OWL API which will parse the input ontology and return results based on selected method. The results could be any feature of the input ontology based on the method that is called.

Dirsumilli, R. and Mossakowski, T.
RESTful Encapsulation of OWL API.
DOI: 10.5220/0005987201500157
In *Proceedings of the 5th International Conference on Data Management Technologies and Applications (DATA 2016)*, pages 150-157
ISBN: 978-989-758-193-9

2 OWL2

OWL2 Web Ontology Language by definition is, “an ontological language for the semantic web with formally defined meaning” (W3C, 2012). Similar to OWL1, OWL2 is also designed to make ontology development and sharing via web easier.

2.1 OWL2 Ontologies

The OWL2 structural specification document contains the conceptual structure of ontologies defined in it. Also the functional style syntax that follows closely the structural specification and allows OWL2 ontologies to be written in compact form is defined. The OWL2 ontologies have the possibility to be viewed as an RDF graph. This relationship of viewing OWL2 as an RDF graph is specified by "mapping to RDF graphs" document.

RDF/XML is the primary exchange syntax for OWL2 which is the only syntax that must be supported by all tools. Although RDF/XML provides interoperability amongst OWL2 tools there exists other concrete syntaxes which can also be considered. Turtle is one such syntax which is based on RDF, OWL2XML is another syntax which uses XML serialization, and Manchester syntax, an editing tool with clearly readable syntax used by several ontology editors. Besides its major purpose that is, to specify the structure of the language, the functional style syntax can also be considered for serialization.

2.2 Entities

The names of classes, properties, datatypes and individuals are represented as IRIs in OWL2 and are collectively known as entities. The OWL2 ontologies are built on top of such entities and datatypes.

The set of entities that occur in ontology are referred as signature in DL.

2.3 Semantics

Although the abstract structure of OWL2 ontologies is defined in OWL2 Structural Specification document, their meaning is not defined. The OWL2 Direct Semantics and OWL2 RDF-Based Semantics are two different ways of assigning meaning to OWL2 ontologies along with a theorem that corresponds to provide a link between them. The reasoners and other OWL2 tools use these two semantics.

A meaning is assigned directly to RDF graphs by the RDF-based Semantics which leads to indirect structuring of ontologies via mapping to RDF graphs. The RDF-based semantics can be directly applied to any OWL2 ontology without any existing restrictions.

A meaning is assigned directly to ontology structures in Direct Semantics. This results in a semantics which are compatible to the model theoretic semantics. Direct Semantics of OWL2 is based on DL, therefore it supports the concepts of DL semantics.

Interpretations: Before discussing the semantics of DL, basic knowledge of "what are interpretations?" plays a major role in understanding the semantics. An Interpretation I consists of a set Δ^I called the domain of I and an interpretation function. ^{I} that maps each atomic class/concept A to set

$$A^I \subseteq \Delta^I$$

each atomic property/Role R to a binary relation,

$$R^I \subseteq \Delta^I * \Delta^I \text{ and}$$

each individual name a to an element,

$$a^I \in \Delta^I$$

(Krötzsch et al., 2012).

The interpretation of basic entities is the base of interpretation of complex concepts and roles.

Logical Consequences: An axiom which holds in all the interpretations that satisfy the ontology is a Logical consequence. As the number of axioms in ontology increases, the number of interpretations that could satisfy the axioms decreases. Contrarily, more number of Logical Consequences follow from fewer interpretations.

Inconsistent Ontology or Unsatisfiable Class: In situations, where no interpretation can satisfy the axioms in an ontology the ontology is considered as inconsistent ontology.

Conservative Extension: This can be explained by considering two ontologies T^1 and T^2. The ontology generated by merging these two ontologies that is, $T^1 \cup T^2$ can be considered as conservative extension of T^1 , w.r.t. a signature S only if every S-consequence of $T^1 \cup T^2$, is already a consequence of T^1. In other words, the merged ontology $T^1 \cup T^2$ is an S-conservative extension of T^1 , if T^1 doesn't change even after adding T^2 to it, as far as consequences over S are concerned.

Module Extraction: The concept of module extraction is, extracting a set of axioms from the ontology, which are relevant to the signature specified. Considering an ontology *O,* and module *M,* then $M \subseteq O$ is a module of *O* with respect to Signature *S* if *O* is an *S*-conservative extension of *M.*

2.4 Profiles

The profiles of OWL2 can be termed as the sublanguages which offer important benefits in particular applications. There are three different profiles defined in OWL2: OWL2 EL, OWL2 QL and OWL2 RL. Each of these profiles is defined as a syntactic restriction over the OWL2 Structural Specification. All these profiles trade various aspects of OWL's expressive power in exchange to different computational benefits.

OWL2 EL: Polynomial time algorithms are enabled for all standard reasoning tasks.

OWL2 QL: Conjunctive queries are enabled to be answered in log space (AC^0) by using standard relational database technologies.

OWL2 RL: Implementation of polynomial time reasoning algorithms is enabled using rule extended database technologies that are operated directly on RDF triples.

3 OWL API

Originally, the OWL API provides a suite of interfaces in addition to a reference implementation which facilitates the use of OWL in a wide variety of applications. A set of inference engines are present at the core of OWL API for manipulating, reasoning and inspection with OWL ontologies. Further, loading and saving of ontologies in varieties of syntaxes is supported by OWL2. Nevertheless, none of the model interfaces in the API are biased or reflects any particular concrete model or syntax. The OWL API design is directly based on the OWL2 Structural Specification. This results in the ontology simply being viewed as a set of annotations and axioms. The names and hierarchies of interfaces for entities, axioms and class expressions in the OWL API closely correspond to the structural specification. Presently, there is nearly a direct relationship between the core OWL API model interfaces and the OWL2 Structural Specification, which means that the high level OWL2 specifications can be directly related to the design of the API.

The OWLOntology interface provides access to efficiently obtain the axioms that are contained within an ontology. The method of storing axioms is provided by different implementations of the OWLOntology interface.

3.1 Parsing and Rendering OWL Ontologies

The concept of aligning OWL API with the structural specifications of OWL2 derives a major benefit of no commitment to a single syntax. Despite of single syntax such as RDF/XML exists which the OWL implementation supports, there exists many other syntaxes which are optimized for various purposes. An out of the box support is included in the OWL API for the purpose of reading and writing ontologies in various syntaxes, which also includes RDF/XML, OWL/XML, Turtle, The Manchester OWL Syntax , KRSS Syntax, OBO flat file format and the OWL Functional Syntax. A registry of parsers and renderers which makes it easy for the OWL API to add support to custom syntaxes, are used by its reference implementation. When the ontology is loaded at the run time, the appropriate parser is automatically selected. Also, the ontologies are saved back after the parsing in the same format from which they were parsed by default, but there exists possibility of converting the ontologies such that they perform syntax conversion tasks like a "save as" operation in editors.

3.2 Reasoner Interfaces

Reasoning is one of the most interesting elements while working with the OWL ontologies. In general, the purpose of the reasoners is to

- check the consistency of the ontologies,
- check for any unsatisfiable classes existing in signature of an ontology, class and property hierarchy computations, and
- Check the entailment Logical Consequences.

To support the interaction between OWL reasoners and OWL API, there exist various interfaces. OWLReasoner is the major interface that provides the methods to perform the previously mentioned tasks.

3.3 Profile Validation in OWL API

The OWL2 specifications provide various profiles, which in turn corresponds to syntactic subsets of the OWL2 language. The profiles that are defined as the OWL2 profiles in the document are the OWL2 EL, OWL2 QL and OWL2 RL.

A complete programmatic access by client software is allowed by profile API, along with fine grained objects which represents particular reasons for profile violations. Also, there exists an web based profile validator which performs the profile validation on an ontology and its imports closure was written with the OWL API and is available in (McGuinness et al., 2007).

4 RESTful SERVICES

4.1 What Is REST?

REST is the 'Representational State Transfer', a concept described by Roy Fielding about the web's architectural style. This web's architectural style is framed with constraints which are grouped into six categories. They are:

- Client-Server
- Uniform Interface
- Layered System
- Cache
- Stateless
- Code-on-demand (Fredrich, 2012).

4.2 RESTful API

A web API which is built on the REST architectural style is the RESTAPI.

URIs
Uniform Resource Identifiers (URIs) are used in REST APIs to address resources. URIs in the present web range from masterwork that transparently communicate the API's resource model such as-
http://www.library.com/books/harry-potter/first-series

Compared to those that is much harder for users to understand such as-
http://www.library.com/48gg0-h9p6-1sfc-7823a7754f94

***http* Methods for RESTful Services:**
A major part of the 'uniform interface' constraint constitutes of *http* verbs. It also provides the action counterpart to the noun-based resource. POST, GET, PUT ad DELETE are the most commonly used and primary *http* verbs. These verbs represent create, read, update and delete (CRUD) operations respectively. There are other *http* verbs such as OPTIONS and HEAD present, but are not frequently used.

GET: It is the verb used to 'read' or 'retrieve' a representation of a resource. An XML or JSON representation and a HTTP response code 200 (OK) is returned by GET in a non-error path. The code 404 (NOT FOUND) or 400 (BAD REQUEST) is returned in case of error path.
Example: GET *http://www.library.com/listbooks*
This retrieves the data based on the arguments passed, which is it lists the books present in the library.

POST: It is the verb used to 'create' new resources. POST particularly creates dependent resources, i.e., a dependent to some other resource. On creating successfully a HTTP return status 201 along with a location header with link to the newly created resource is returned.
Example:
POST *http://www.library.com/newbook/harry-potter/second-series*
A new book harry-potter second-series is added, that is new resource is created.

5 RELATED TOOLS

Plenty number of researches have been and are still working with OWL API, to utilize its features for the semantic web. (Bergman, 2011) lists the available tools built on top of OWL API. Few of these tools are related to the current developed tool. Hence, we take a look through these related tools.

Protégé: Protégé is an Ontology editor that is built on top of OWL API and completely supports OWL2 Web Ontology Language (Musen, 2015).

WebProtégé: WebProtégé was built as a “collaborative web-based platform that supports ontology editing and knowledge acquisition, and that can be easily tailored for domain-expert use” (Tudorache et al., 2013).

OWLTools:"OWLTools leverages the full features of OWL API and OWL Reasoner API, but provides convenience methods for common tasks" (“OWL Tools,” 2014).

OWL API Wrapper: The OWL API Wrapper (Manuel, 2015) similar to OWL Tools is a command-line utility. This tool wraps the OWL API such that it can parse OBO, OWL and RDFS ontologies.

OWL Toolkit: From (Xiao, 2015), the OWL Toolkit is also a command-line utility built on top of OWL API.

Web VOWL: "WebVOWL (Web-based Visualization of Ontologies) is a web application for user-oriented visualization of ontologies“ (Steffen, 2015).

Ontodev.owlapi: This tool provides a thin Clojure wrap around the OWLAPI and other utilities such as, Hermit reasoners, for working with RDF/XML representations of OWL ontologies.

Apache Stanbol: Ontology developers can use the components Apache Stanbol Ontology Manager, Apache Stanbol Reasoner and any other required service individually through the provided RESTful web service.

Comparison of Apache Stanbol to current RESTful OWLAPI

The components of Apache Stanbol that deal with ontologies are designed towards individual ontologies and also ontology networks. Nevertheless, all the components are built as RESTful web services, which is similar to the current work. Hence using reasoner services of Apache Stanbol has few similarities as using the RESTful OWLAPI.

- The Apache Stanbol Reasoners allows manual selection amongst different types of reasoners that is rdfs, owl, HermiT etc.. RESTful OWLAPI uses OWLAPI along with OWL Reasoner and additionally HermiT Reasoner. But, the selection of Reasoners in RESTful OWLAPI is not manual.
- Apache Stanbol Reasoners are restricted to only three methods, where as the RESTful OWLAPI has wide methods described which deals not only with reasoners, but also OntologyManagers, DataFactory and other methods of OWLAPI
- The base of the modules of Apache Stanbol Reasoner includes OWLAPI and Jena abstract services. The RESTful OWLAPI is complete encapsulation of OWLAPI alone.
- The consistency checker in Stanbol can only verify the consistency of the input ontologies, but cannot yet return any explanations of inconsistency. The OWLAPI already implements this process and is encapsulated in the RESTful OWLAPI.
- The Apache Stanbol Ontology Manager is built to work with ontologies and also ontology networks. This allows the manager to control a whole stream of ontologies. This is not possible in the RESTful OWLAPI, as it is designed to only work with a single input ontology at a time.
- A complete access through all the ontologies stored in the Stanbol persistence layer or any input ontology library is provided to the Apache Stanbol Ontology Manager. The RESTful OWLAPI can manage the input ontology and no possibilities to hold on running ontologies like in Apache Stanbol.

6 RESTful OWL API

RESTful OWL API is a web service to access the methods of OWL API. The major goal while building this tool is not only to wrap the selected java methods of OWL API, but to enable the web ontology developers to access these methods through a web interface. Hence, the concepts of REST are used to build the web service through which all the protocols are passed. Although OWL API supports creation, manipulation and serialization of OWL ontologies along with reasoning over ontologies, the RESTful OWLAPI only encapsulates the methods which do not make any changes in the original ontology. The restrictions on encapsulating methods are framed such that the tool should support open ontology repositories and other web services that use ontologies and should not stand as a web ontology editor. Therefore the methods which include parsing and reasoning over ontologies are the major focus in this work.

6.1 Resource Identifier

One of the major constraints of REST design architectural style is the ability of global addressing through resource identifiers. HTTP methods required by REST design are addressed by Uniform Resource Identifier (URI). A URI provides simple and extensible means for identifying a resource (Berners and Fielding, 2013).
Example: http://www.library.com/listbooks

Along with resource identification, the identifiers in the URI can also represent the input arguments. Therefore, from the above example, "http://www.library.com" is the resource identifier, followed by argument *listbooks.* Similarly in the RESTful OWL API, resource identifier used is in format:
http://www.example.server.com/:methodName/:coded_IRI/:parameters/

In the above URI which is the format followed for RESTful OWL API, identifiers indicate-
:example.server - Example server name of the resource, which is http://owlapi.hets.eu/

:method – The method to be implemented over the input ontology.
:coded_iri – The input OWL ontology has to be ontology from a web ontology repository like Ontohub. The URI that identifies the ontology should be converted into IRI (discussed in the following section) such that it fits to be identified as a single argument.
:parameters – Not all the methods in the RESTful OWL API have additional parameters passed. Therefore it doesn't stand to be a compulsory attribute. Nonetheless, it has to be passed when demanded by the method.

IRI
Internationalized Resource Identifier (IRI) is a new protocol designed as an extension of URI, with a wide repository of characters. In general, IRI is the encoded URI and they can be mapped to URIs and vice versa, hence in case of resource identification, IRIs can be used in place of URIs. In RESTful OWL API, IRI is used to identify input OWL ontology instead of URI, as that it can be pointed as a single argument. Therefore, while passing the input ontology, the URI of OWL ontology must be first converted to IRI; it can be done using http://www.url-encode-decode.com/.
Example: when encoded the URI given above, it returns,
http%3A%2F%2Fwww.url-encode-decode.com%2F

6.2 HTTP Methods

As discussed earlier, HTTP verbs stand to be major asset for web services to attain „uniform interface“. Although HTTP methods offer GET, POST, PUT and DELETE, which intend to 'read', 'create', 'update' and 'delete' respectively, they are not all used in the current tool. RESTful OWL API utilizes the methods „GET“ and „POST“ which do not aim at rendering the input OWL ontology.

6.3 Accept Headers

Much elegant approach while designing a RESTful web interface is to include HTTP accept headers that supports the output format. These headers are to be implemented specifically for custom result and does not have any specific format or standards to follow. To indicate that the request is limited to a particular set of desired type, accept headers point towards the format of output. For example, "Accept: text/json" indicates that the result with a set of JSON text in is only returned.

RESTful OWL API accepts headers with JSON, HTML or PLAIN (in few methods).
Syntax:
```
  Accept        = "Accept:"
                    #( media-range [ accept-params ] )
      media-range   = "text"
      accept-params    = "JSON" or "HTML" or
"PLAIN" or "XML"
```
Example:
```
curl -X „Accept: text/JSON“ POST
http://owlapi.hets.eu/modularity/http%3A%2F%2Fo
ntohub.org%2Ftones%2Fwww.co-
ode.org%2Fpizza.owl/hasSpiciness/manowl
```

6.4 RESTful OWL API Design

In this section, we discuss in detail the procedures followed along with input, outputs and examples on how to use each of the methods.

Design:
The core of RESTful OWL API is the encapsulation of the selected java methods of OWL API. Additionally, to meet the RESTful interface constraints, the URIs act as the source of input. Arguments including OWL ontology IRI, method name and any parameters required are passed through the URI as input. Once the encapsulation is achieved for the given input, the results are rendered mostly into JSON and in some scenarios into XML and returned as output. An overview of this design can be seen in Figure below.

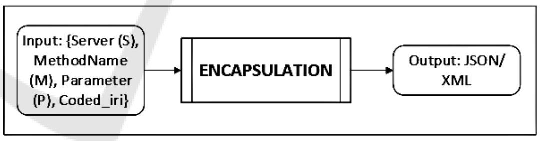
Input: {Server (S), MethodName (M), Parameter (P), Coded_iri}
ENCAPSULATION
Output: JSON/ XML

Figure 1: An overview of RESTful OWL API.

Input:
Server (s) is http://owlapi.hets.eu, provided.
Method Name (M) refers to the method to be performed and Parameter P depends on M.
Coded_IRI is the OWL Ontology URI that is encoded into IRI.

Encapsulation:
Encapsulation here refers to the process of wrapping java methods in the OWL API to reproduce their functionalities in the form of a RESTful web service. In this section, we clearly discuss the methods encapsulated along with procedures followed.

In the Encapsulation module, firstly the OWL Ontology is extracted from the input IRI using OWL API method. Then, method is encapsulated based on the method name that is selected. If the parameters passed meet the requirements of the method, and then the result is returned by parsing the input ontology based on method and parameters if any. Following are the various methods that are present in the RESTful OWL API.

The documentation and code to work with RESTful OWL API can be found at,
https://github.com/ramyadirsumilli/rest_owl_api
And, it can be accessed through,
http://owlapi.hets.eu/.
Methods of RESTful OWL API include:

ExploreClasses: Retrieves list of OWL Classes from input ontology.

Descendants: List of direct subclasses of each OWL Class is retrieved from input ontology.

ConvertOntology: Converts format of input ontology to any required format.

InferredOntology: An ontology is generated based on inferred axioms in the input ontology.

Hierarchy: Hierarchy of the input ontology is retrieved.

Modularity: Module ontology from the input ontology based on input signature is generated.

PetInstances: Instances of OWL Classes along with their properties are retrieved.

CheckProfiles: Input ontology is verified against all the OWL Profiles.

VerifyProfile: Input ontology is verified against a specified OWL Profile.

Explanations: Satisfiable and Unsatisfiable OWL Classes along with explanations for unsatisfiability in the input ontology are listed.

UnsatisfiedClassExplanation: Single input OWL Class is verified for satisfiability and returns explanations in case of Unsatisfiable OWL Class.

LookupRestrictions: Restrictions if any present on the input OWL Class are retrieved.

ReadAnnotations: The annotations present in the input ontology in specified language are retrieved.

LogicalConsequences: retrieves logical consequences from the ontology.

7 APPLICATIONS

The motivation behind designing RESTful OWL API is to make the methods of OWL API available for ontology repositories and also for other applications that are not written in Java, such that they can manage, validate and know the ontologies.

7.1 Ontohub

Ontohub is an open ontology repository for managing distributed heterogeneous ontologies. That is, it supports in organizing, collection, retrieval, mapping, development, evaluation and translation of huge array of ontologies. Its distributed nature makes it possible for communities to share and exchange contributions easily. Additionally, its heterogeneous nature enables integration of ontologies that are formulated in various ontological languages. Ontohub is an open source ontological repository, where users can browse, upload, search and annotate basic ontologies in several languages using a web frontend.

Using the RESTful OWL API, the ontologies in the Open Ontology Repository (OOR) Ontohub can be parsed and exhibit features of OWL API, such as verification of consistency of a ontology, modularity, inference and so on. An added advantage while using RESTful OWL API with Ontohub is that, it doesn't intend at making any changes in the ontologies uploaded in repository. But, the ontologies in the repositories can work and use all the functionalities exhibited by the RESTfully Encapsulated OWL API.

7.2 Others

The RESTful OWL API exhibits most of the features of OWL API in the form of a RESTful web service. Therefore, any ontological repository or in that case any ontology based web application that requires validation or any other functionality of OWL API can use the RESTful OWL API.

For instance the methods of RESTful OWL API can be used in ontology matching, which plays a major role in ontology integration. There are many approaches for ontology matching, some require module extraction as in (Solimando et al., 2014) and few also depend on logical consequences such as (Jiménez-Ruiz et al., 2009). While working with any of these approaches in the form of an web service, the RESTful OWL API could be called easily from the web service instead of OWL API which decreases the efforts of calling an java based API.

8 SUMMARY

RESTfully Encapsulated OWL API, a tool designed to allow the access of OWL API methods through a web service. The tool has been implemented to provide the following features:

- **RESTful** - It follows the constraints of RESTful web architecture to act as a web API.
- **Standalone** - Although the core proposal of the tool was to support the Ontohub, this service along with providing support to Ontohub, also stands as an independent framework. Therefore, any service that deals with ontologies on the web can utilize the functionalities of RESTful OWL API.
- **Wrap OWL API** - All the methods in the OWL API that support working with a single input ontology are encapsulated. That is, all the java methods, which help in understanding an input ontology by parsing through it and retrieving data from the inputs by making no changes in the core ontology are wrapped.

The tool helps in exploring ontology and retrieving data such as, list of classes, or direct sub classes or even the pet instances present in the input ontology. Also, the reasoning services are supported by this tool. Therefore, information like the reasons for inconsistent ontology or explanations regarding single class unsatisfiability can also be retrieved. Additionally, the tool supports the methods inference and modularity, which are key concepts of the OWL API. These methods return a new ontology based on the input ontology and parameters passed along with it. Verification over a profile validation is an added advantage, the tool along with cross verifying input ontology with the OWL Profiles, also returns a profile report that is generated by wrapping the respective methods of OWL API.

Along with a support for ontology management in open ontology repositories like Ontohub, the RESTful OWL API stands to be a web service offering functionalities of OWL API. Therefore, any web developer or researcher using ontologies and interested in working with OWL API as an interface can use this tool (RESTful OWL API), and work on top of it.

REFERENCES

Bergman, M. K. (2011). Thirty OWL API Tools. Retrieved February 16, 2016, from http://www.mkbergman.com/977/thirty-owl-api-tools/

Berners, T., & Fielding, R. (2013). Uniform Resource Identifier (URI): Generic Syntax Status. *Journal of Chemical Information and Modeling*, *53*, 1689–1699. http://doi.org/10.1017/CBO9781107415324.004.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., & Orchard, D. (2004). Web Services Architecture. *Group*, *22*(February), 19–26. http://doi.org/10.1023/B:BTTJ.0000015492.03732.a6.

Fredrich, T. (2012). RESTful Service Best Practices Recommendations for Creating Web Services, 1–25.

Jiménez-Ruiz, E., Cuenca Grau, B., Berlanga, R., & Horrocks, I. (2009). Towards a logic-based assessment of the compatibility of UMLS sources. *CEUR Workshop Proceedings*, *559*, 2–5. http://doi.org/ 10.1186/2041-1480-2-S1-S2.

Krötzsch, M., Simancik,1 F., & Horrocks, I. (2012). A description logic primer. *arXiv Preprint arXiv:1201.4089*, (June), 1–17. Retrieved from http://arxiv.org/abs/1201.4089.

Manuel, S. (2015). OWL API Wrapper. Retrieved February 16, 2016, from https://github.com/ncbo/ owlapi_wrapper.

McGuinness, D., Fox, P., Cinquini, L., West, P., Benedict, J., & Garcia, J. (2007). Current and future uses of OWL for Earth and Space science data frameworks: Successes and limitations. *CEUR Workshop Proceedings*, *258*.

Musen, M. A. (2015). The Protégé Project. *AI Matters*, *1*(4), 4–12. http://doi.org/10.1145/2757001.2757003.

OWL Tools. (2014). Retrieved February 16, 2016, from https://github.com/owlcollab/owltools/wiki/OWLTools-Intro.

Solimando, A., Jim, E., & Guerrini, G. (2014). Detecting and Correcting Conservativity Principle Violations in Ontology-to-Ontology Mappings, 1–16.

Steffen, L. (2015). WebVOWL: Web-Based Visualization of Ontologies. *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8982*, 225–232. http://doi.org/10.1007/978-3-319-17966-7.

Tudorache, T., Nyulas, C., Noy, N. F., & Musen, M. A. (2013). WebProtégé: A collaborative ontology editor and knowledge acquisition tool for the web. *Semantic Web*, *4*(1), 89–99. http://doi.org/10.3233/SW-2012-0057.

W3C, O. W. L. W. group. (2012). OWL 2 Web Ontology Language Document Overview (Second Edition), (December 2012), 1–8. Retrieved from http://www.w3.org/TR/owl2-overview/

Xiao, G. (2015). OWL Toolkit. Retrieved February 16, 2016, from https://github.com/ghxiao/owl-toolkit.