ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 6, Issue 2, February 2017

Semantic web based Inference capabilities using Jena framework

Asad Ali,
Department of Computer Engineering,
Near East University, North Cyprus via Mersin 10 Turkey

Attiq Ur Rahman
Department of Computer Engineering,
Near East University, North Cyprus via Mersin 10 Turkey

Abstract: Development of a semantic web applications require rules and rule based inference engine in addition to tradional semantic web tools like RDF, OWL, and SPARQL. Rules are used to infer new knowledge based on the one already exist in the knowledge base/ontology and can be added as RDF triples. The rules are fired by the reasoner which can be used and activated in the application. Apache Jena allows us to infer new knowledge based on the rules and reasoners it support and provides. This paper uses a short ontology developed in Protege and uses various Jena rules to generate/infer new knowledge and triples.

KeyWords: Semantic Web. Jena, Inference, Reasoning

1. **Introduction:** In Semantic Web, Ontologies are the core elements and can be think of in two ways:

- Rdf Schema: Which extends RDF and provides subClassOf, subPropertyOf, domain and range for the description of ontologies.
- OWL: Web Ontology Language (OWL) further extends RDF schema and provide basis for Description Logics and define three different sub-languages:
  - ❖ OWL-Lite: Minimal support like cordinality
  - ❖ OWL DL: More expressiveness , based on Description Logics
  - ❖ OWL Full: minimal RDFS compatibility and computationally expensive.

2. **Semantic Web Reasoners:** There are some semantic web based reasoners, described as:

- Notation3: It is also called N3 for short, and is considered humman readable. It supports a reasoning engine CWM, written in Python and is open source.
- (Renamed ABox and Concept Expression Reasoner (RACER): An OWL based reasoner which allows ontology based query answering system based on our data.
- Mandarax: It provides backward reasoning capabilities to deduce new knowledge. It is open source and object oriented.
- Jena reasoner: It provides support for Jena based rules and hence most widely used for building inference based semantic web applications.

3. **Jena Rules:** Apache Jena is a Java API for building semantically rich applications by providing a programmatic environment for RDF, RDFS, OWL, and SPARQL[4].

Jena API provides a rule based inference engine which deduce knowledge using variety of Jena rules. Thus if we have an ontology or knowledge base, we can easily make inference on it using some Jena rules and reasoner.

3.1: Jena Reasoners: Jena can provide various reasoners inside the application based on user needs and requirements. Jena mainly support RDFS and OWL reasoning but also have support of Generic reasoner.

The structure of the Jena reasoner is shown in figure 1.

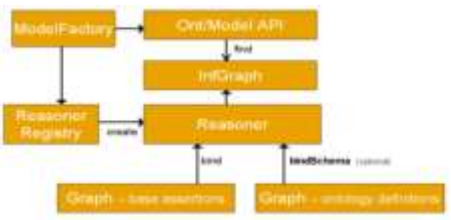

Figure 1

We can discuss various Jena reasoners below:

- RDFS Rule Reasoner: It is used to infer new knowledge from static information and implements RDFS entailments.
- OWL Rule Reasoner: OWL, OWL Mini, OWL Micro reasoners can be applied to data to infer and produce knowledge.
- Transitive Rule Reasoner: It provides support to infer rdfs:subClassOf and rdfs:subPropertyOf .
- Generic Rule Reasoner: If the above reasoners does not work for user needs, Jena provides a Generic reasoner which uses user defined rules and can be executed as forward chaining, backward chaining and hybrid.

Inside Jena code, each of the above reasoner is implemented as the instance of ReasonerFactory class which uses its create() method to configure reasoner. Once created, the ontology data file is then passed to it.

4. **Proposed work**: This paper uses a small ontology, Research.owl, created in Protege which contain classes as shown in figure 3. Student Class has sublasses Master and Phd. Master Class in turn has “researchmaster and “taughtmaster” subclasses.
A Phd student has type both Expert and Student because Phd student can also have some researches.
If we try to query Expert class, it will give us instances of both Student class(Phd class) and Expert class. However, based on Jena generic rules, we will filter Expert class instances to Phd class instance and using inference rules to assign Phd class instances to a class “StudentExpert”. StudentExpert class has no individuals/instances hard coded into it, rather will be created based on Jena user defined rules.

The ontology has object properties:

- hasResearch
- associatedWith
- familiarWith
- ResearchOf

For example, Expert hasResearch Research, Research associatedWith Subject, Expert familiarWith Subject. ResearchOf is the inverse property of hasResearch.

Figure 2

We will use the following Jena rules to deduce new knowledge:

- Transitive rule: Which says if ExpertA hasResearch ResearchA and ResearchA associatedWith SubjectB, then we can infer ExpertA familiarWith SubjectB. There is no instance of “familiarWith” property and we will assign instance to this using inference.
- OWL Inverse rule: There is an object property ResearchOf which is the inverse property of hasResearch and instance of this property can also be get from inference.
- RDFS Subclass/Superclass relationship: Query inside Protege will just answer immediate subclasses. For instance, in the proposed work, Protege will only show Master and Phd as subclasses and will not display subclasses of Master class (researchmaster/taughtmaster). We will achieve it using OntModelSpec.OWL_MEM_RDFS_INF - that will use the default RDFS inference configuration.
- Generic rule: we also define rule (user defined) as generic rulein which instance is provided to StudentExpert class based on this rule.

4.1: **Transitive Rule**: First we will use Jena transitive rule if X likes Y and Y likes Z, then X likes Z.
In proposed work, we have “hasResearch” which means an Expert class individual have some research i-e Dr Melike hasResearch Ontologies.

There is "associatedWith" object property which says that Research is associated with the subject, for instance, Semantic Web. And we can finally infer that Dr Melike(Expert class instance) familiarWith Semantic Web(Subject class individual). There is no instance of "familiarWith" property but we will infer it using the rules below:

[rule1:(?x
http://www.semanticweb.org/asadali#hasResearch
?y) " +
"(?y
http://www.semanticweb.org/asadali#associatedWith ?z +
" ->(?x
http://www.semanticweb.org/asadali/#familiarWith
?z)]";

4.2. **OWL Inverse**: This rule can be used as: If Expert A hasResearch Research A, then Research A is the ResearchOf ExpertA. Property "ResearchOf" has no instance but will be created using this rule.

[rule2:(?x
http://www.semanticweb.org/asadali#hasResearch
?y) " +

➔ (?y
http://www.semanticweb.org/asadali#ResearchOf
?x) "

Figure 3 shows Reseach have Experts

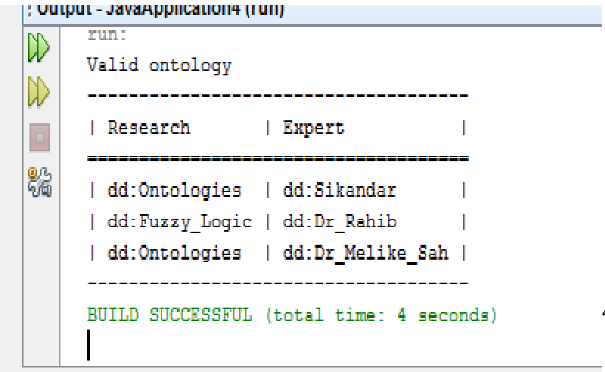

Figure3

4.3. **Jena Generic Rules**: These are user defined and widely used rules which users implement according to their needs and requirements.
In our proposed ontology, we have some experts which have some researches but the class Phd also has the type of Expert (along with type Student class) and thus also have some researches. So the Expert class query will return the following instances:

- Dr Melike Sah
- Dr Rahib
- Sikandar
- Ali

The first two are from Expert class and instance Sikandar & Ali from Phd class. We will use Jena rules to filter instances from Phd class and then assign them to a new class "StudentExpert" which has no instances but will be generated using Jena generic rules (Forward chaining). Rules are:

"[rule3:(?x
http://www.semanticweb.org/asadali#hasResearch
?y) " +
"(?x http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.semanticweb.org/asadali#Student)" +

"->(?x http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.semanticweb.org/asadali#StudentExpert)]";

The query used here is:

Select ?StudentExpert " + "where {
?StudentExpert rdf:type dd:StudentExpert }

Figure 4

4.4. **RDFS Subclass/Superclass relationship**: Executed inside Protege, the "rdfs:subClassOf" return only immediate subclasses i-e if we try to query the subclass of Student in Protege, it will return only Master and Phd as shown in figure 5. It does not shows the subclasses of Master class i-e ResearchMaster and TaughtMaster.

This is because SPARQL doesn't understand the subclass assertions, if we have to get the full list of Student subclass, we will use Jena methods and will pass the "OntModelSpec.OWL_MEM_MICRO_RULE_INF" to the OntModel which will then do the rest automatocally.The figure 6 shows the detailed subclasses of Student using Jena methods.

Figure 5

Figure 6

5. **Conclusion:**Reasnoning provides vital role in semantic web applications, particularly when the applications are large and real life. Infact the main difference between tradional web and semantic web is that the later provides much more explicit reasning and inference capabilities to our application.

6. **References:**

[1]. Luo Zhong, et al "
The Jena-Based Ontology Model Inference and Retrieval Application" July 2012.

[2]. Rewat Lerlertvanich, et al, "Facade Layer for Apache JENA" ARPN Journal of Systems and Software.

[3]. Ayesha Ameen, et al "Reasoning in Semantic Web Using Jena" Computer Engineering and Intelligent Systems www.iiste.org ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online) Vol.5, No.4, 2014

[4]. Jeremy J. Carroll, et al, "Jena: Implementing the Semantic Web Recommendations"

[5]. Kruti Jani, et al, "A Study on Semantic Web Framework: JENA and Protégé" Volume : 4 | Issue : 1 | Jan 2014 | ISSN - 2249-555X

[6]. Michael Grobe, "RDF, Jena, SparQL and the "Semantic Web" Indiana University Indianapolis, Indiana USA