SEMANTIC TECHNOLOGY PROJECT REPORT

PROTEGE MAPPING VALIDATOR

February 2, 2017

Student ID: 13396
Student Name: Aman Sinha
Free University of Bozen, Bolzano
European Masters in Computational Logic

Contents

OBJECTIVE

The objective of this project is to build a mapping validator in Protege (a plugin for Ontop) in order to verify if the target query (we call this as mapping) is written correctly. If the target query is written correctly, then, the generated triples are shown. Otherwise, an error message should be displayed stating that the mapping is incorrect.

MOTIVATION

Currently, the mapping dialog panel in Protege only validates the SQL query and gives the functionality of testing SQL query. We would like to extend similar functionality to test the mapping. This would help the users to check if their mapping is correct.

INTRODUCTION

Ontop is a platform to query relational databases as Virtual RDF Graphs using SPARQL. The homepage for Ontop is [1]. In order to acquire in depth knowledge about Ontop, refer [2]. Whereas, an extensive overview can be found in [3]. Protege is a free, open-source platform that provides a growing user community with a suite of tools to construct domain models and knowledge-based applications with ontologies. The architecture of extensions (e.g. Protege) built over Ontop is explained in figure 1. In this figure, the Quest Core API is at the bottom and OWL API is built over it. The extensions are built over this layer and they use OWL APIs. Ontop Core APIs and OWL APIs are built at UNIBZ, Italy where as the Protege plugin is built at Stanford University, USA.

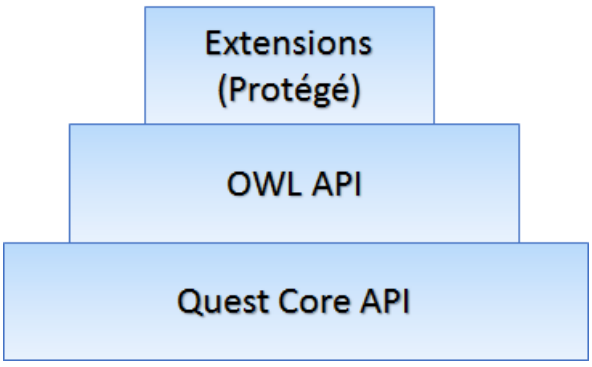

Figure 1: Extension architecture

In this project, we are concerned with the development on Protege plugin. The workflow inside the Protege plugin is shown in figure 2. In this figure, Protege has menus, tabs, reasoner etc. Since we are concerned with the mapping panel, it falls under the Tabs category. These panel use OWL APIs to implement their functionality.

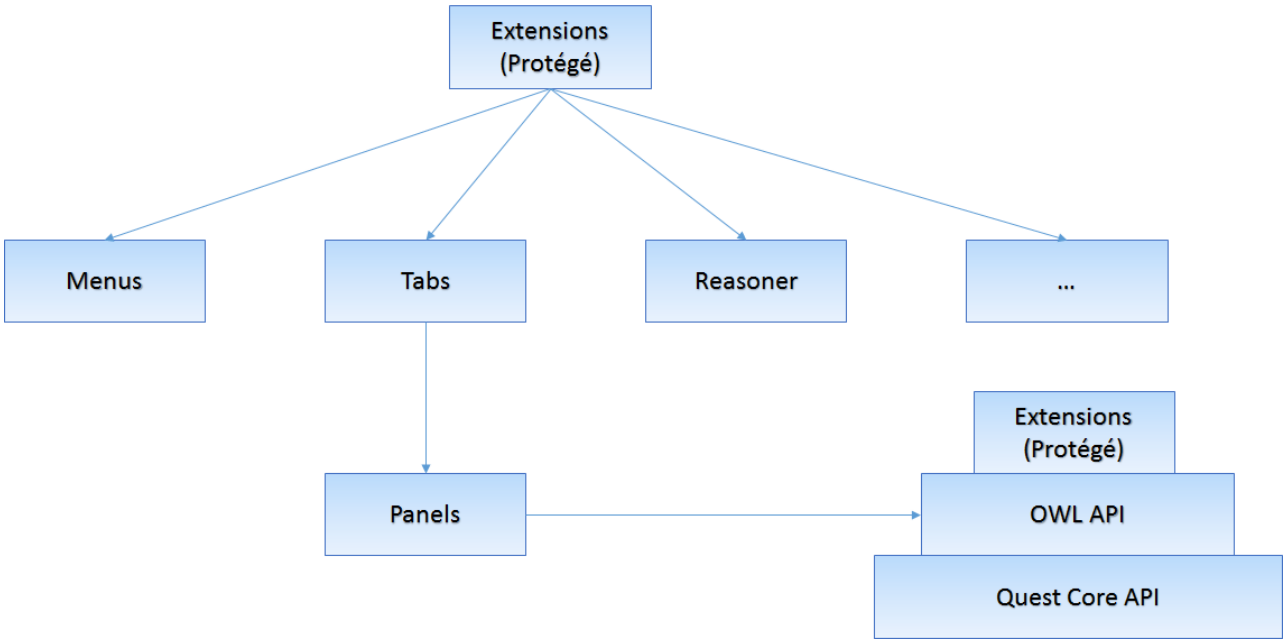

Figure 2: Workflow

Currently, the mapping dialog in Protege looks as in figure 3. In this figure, there is a functionality to test SQL Query. We would like to modify the panel and add a similar button which would check if the mapping is correct. In the figure, the result of the SQL is also shown.

During the development of the feature (mapping validator), the ontology file which is used for testing purpose is available on [4]. The tutorial explains how one can use Protege. One should read the tutorial before proceeding further.

INSTALLATION

The code for the Ontop is available at [5]. From [5], using git, one could fork and clone the repository. In order to debug the OBDAlib Plugin, one could follow the steps mentioned in [6].

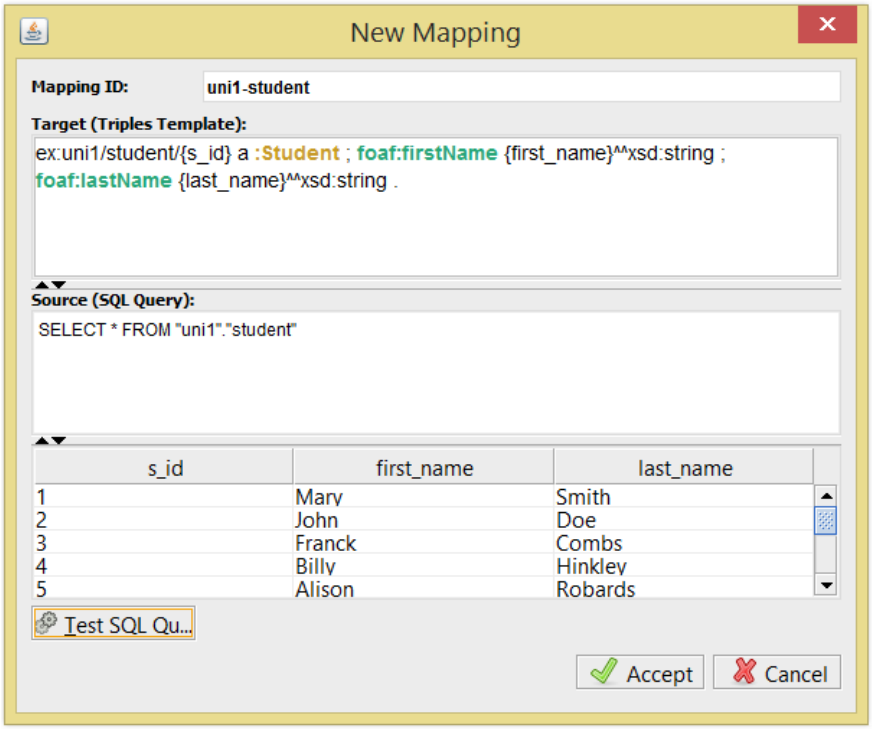

Figure 3: Mapping Dialog Panel

Softwares Used

This section lists the softwares which were used for the development of the feature.

- Netbeans IDE 8.2 - because of historic reasons - in order to view the form file and designing the GUI (panel).
- IntelliJ IDEA Community Edition 2016.2.5
- JRE: 1.8.0_112-release-287-b2 x86
- JVM: OpenJDK Server VM by JetBrains - shipped with IntelliJ IDE
- Git lfs
- Git bash

METHODOLOGY

The steps taken in order to build the mapping validator feature are listed below.

1. Using Netbeans, the mapping dialog was modified and the new GUI looks as in figure 4.

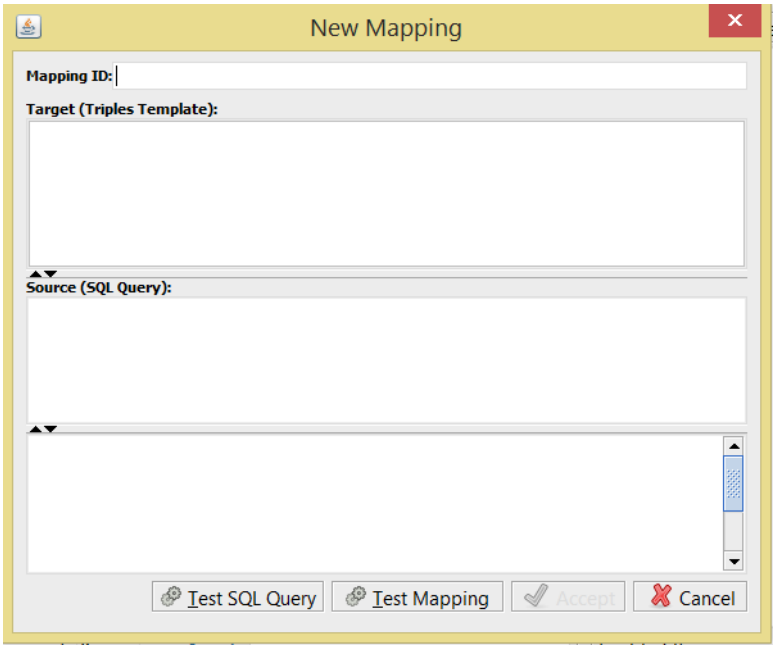

Figure 4: Mapping Dialog Panel

2. In order to implement the functionality of the "Test Mapping" button, the main idea was to attach a reasoner with the button and as soon as the button is clicked, the reasoner should start.

3. Few validation checks are performed in order to check whether the target query and the source are null.

4. Once the validation checks are done, the reasoner is created using the QuestOWL API using createReasoner call. The createReasoner call takes two arguments. The first argument is the ontology and the second argument is the QuestOWLConfiguration which is built from the OBDAModel. The OBDAModel consits of one mapping which is made from the TargetQuery (provided by the user in the textbox). The ontology is taken from the OWLEditorKit which is made to pass through the constructor. The OWLEditorKit has a function to get the current active ontology. Using these, a reasoner is created.

5. If the target query is written correctly and the Test Mapping button is clicked, the triples are displayed (see figure 5).

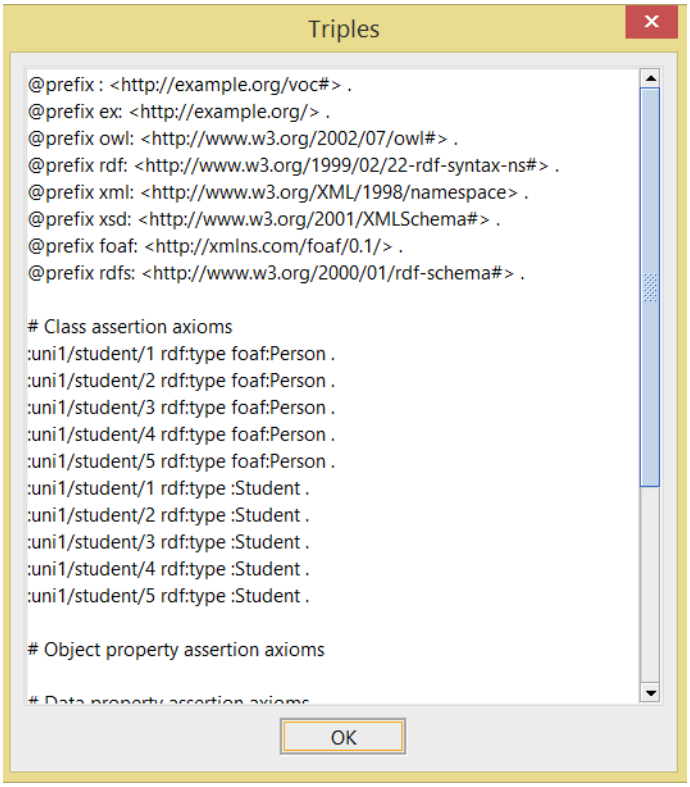

Figure 5: Generated Triples

6. If the target query is incorrect, the error dialog is displayed (see figure 6).

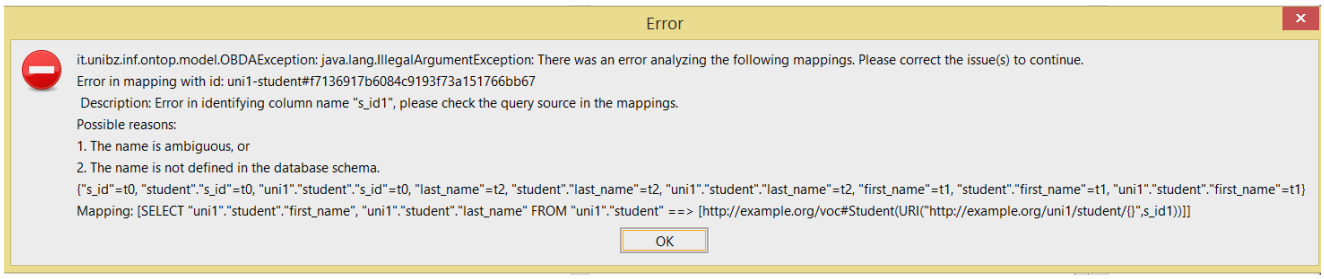

Figure 6: Error

One can also see examples for creating a reason from the OWI API examples available at [7].

CODE

```
1  private boolean mappingValidator(String targetQueryString, String sourceQueryString)
2      {
3          if (targetQueryString.isEmpty()) {
4              JOptionPane.showMessageDialog(this, "ERROR: The target cannot be empty",
       "Error", JOptionPane.ERROR_MESSAGE);
5              return false;
6          }
7          if (sourceQueryString.isEmpty()) {
8              JOptionPane.showMessageDialog(this, "ERROR: The source cannot be empty",
       "Error", JOptionPane.ERROR_MESSAGE);
9              return false;
10         }
11         OBDAModel targetQueryOBDAModel = new OBDAModelImpl();
12         targetQueryOBDAModel.setPrefixManager(obdaModel.getPrefixManager());
13
14         List<Function> targetQuery = parse(targetQueryString);
15         //check which case target query is null
16         if (targetQuery != null) {
17             final boolean isValid = validator.validate(targetQuery);
18             if (isValid) {
19                     URI sourceID = dataSource.getSourceID();
20                     System.out.println(sourceID.toString()+" \n");
21
22                     OBDASQLQuery body = dataFactory.getSQLQuery(sourceQueryString);
23                     System.out.println(body.toString()+" \n");
24
25                     OBDAMappingAxiom newmapping = dataFactory.getRDBMSMappingAxiom(
       txtMappingID.getText().trim(), body, targetQuery);
26                     System.out.println(newmapping.toString()+" \n");
27
28                     targetQueryOBDAModel.addSource(dataSource);
29
30                     targetQueryOBDAModel.addMapping(sourceID, newmapping, true);
31                     try
32                     {
33                         String message = callReasoner(targetQueryOBDAModel);
34                         JTextArea textArea = new JTextArea(message);
35                         JScrollPane scrollPane = new JScrollPane(textArea);
36                         textArea.setLineWrap(true);
37                         textArea.setWrapStyleWord(true);
```

```
38                          scrollPane.setPreferredSize( new Dimension( 500, 500 ) );
39                          JOptionPane.showMessageDialog(this, scrollPane, "Triples",
    JOptionPane.PLAIN_MESSAGE);
40                      }
41                      catch (Exception e)
42                      {
43                          JOptionPane.showMessageDialog(this, e.getMessage(), "Error",
    JOptionPane.ERROR_MESSAGE);
44                      }
45
46
47              } else {
48                  // List of invalid predicates that are found by the validator.
49                  List<String> invalidPredicates = validator.getInvalidPredicates();
50                  String invalidList = "";
51                  for (String predicate : invalidPredicates) {
52                      invalidList += "- " + predicate + "\n";
53                  }
54                  JOptionPane.showMessageDialog(this, "This list of predicates is
    unknown by the ontology: \n" + invalidList, "New Mapping", JOptionPane.
    WARNING_MESSAGE);
55                  return false;
56              }
57          }
58
59          return true;
60      }
```

Listing 1: Mapping Validator Function

```
1  private boolean mappingValidator(String targetQueryString, String sourceQueryString)
2      {
3          if (targetQueryString.isEmpty()) {
4              JOptionPane.showMessageDialog(this, "ERROR: The target cannot be empty",
    "Error", JOptionPane.ERROR_MESSAGE);
5              return false;
6          }
7          if (sourceQueryString.isEmpty()) {
8              JOptionPane.showMessageDialog(this, "ERROR: The source cannot be empty",
    "Error", JOptionPane.ERROR_MESSAGE);
9              return false;
10         }
11         OBDAModel targetQueryOBDAModel = new OBDAModelImpl();
```

```
12          targetQueryOBDAModel.setPrefixManager(obdaModel.getPrefixManager());
13
14          List<Function> targetQuery = parse(targetQueryString);
15          //check which case target query is null
16          if (targetQuery != null) {
17              final boolean isValid = validator.validate(targetQuery);
18              if (isValid) {
19                      URI sourceID = dataSource.getSourceID();
20                      System.out.println(sourceID.toString()+" \n");
21
22                      OBDASQLQuery body = dataFactory.getSQLQuery(sourceQueryString);
23                      System.out.println(body.toString()+" \n");
24
25                      OBDAMappingAxiom newmapping = dataFactory.getRDBMSMappingAxiom(
    txtMappingID.getText().trim(), body, targetQuery);
26                      System.out.println(newmapping.toString()+" \n");
27
28                      targetQueryOBDAModel.addSource(dataSource);
29
30                      targetQueryOBDAModel.addMapping(sourceID, newmapping, true);
31                      try
32                      {
33                          String message = callReasoner(targetQueryOBDAModel);
34                          JTextArea textArea = new JTextArea(message);
35                          JScrollPane scrollPane = new JScrollPane(textArea);
36                          textArea.setLineWrap(true);
37                          textArea.setWrapStyleWord(true);
38                          scrollPane.setPreferredSize( new Dimension( 500, 500 ) );
39                          JOptionPane.showMessageDialog(this, scrollPane, "Triples",
    JOptionPane.PLAIN_MESSAGE);
40                      }
41                      catch (Exception e)
42                      {
43                          JOptionPane.showMessageDialog(this, e.getMessage(), "Error",
    JOptionPane.ERROR_MESSAGE);
44                      }
45
46
47              } else {
48                  // List of invalid predicates that are found by the validator.
49                  List<String> invalidPredicates = validator.getInvalidPredicates();
50                  String invalidList = "";
```

```
51                  for (String predicate : invalidPredicates) {
52                      invalidList += "- " + predicate + "\n";
53                  }
54                  JOptionPane.showMessageDialog(this, "This list of predicates is
    unknown by the ontology: \n" + invalidList, "New Mapping", JOptionPane.
    WARNING_MESSAGE);
55                  return false;
56              }
57          }
58
59          return true;
60      }
```

Listing 2: Mapping Validator Function

Bibliography

[1] *Ontop homepage*. [Online]. Available: http://ontop.inf.unibz.it/ (visited on Feb. 2, 2017).

[2] *Ontop wiki*. [Online]. Available: https://github.com/ontop/ontop/wiki (visited on Feb. 2, 2017).

[3] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, and G. Xiao, "Ontop: Answering SPARQL queries over relational databases", *Semantic Web*, vol. 8, no. 3, pp. 471–487, 2017. DOI: 10.3233/SW-160217. [Online]. Available: http://dx.doi.org/10.3233/SW-160217.

[4] *Ontop examples*. [Online]. Available: https://github.com/ontop/ontop-examples/blob/master/ekaw-tutorial-2016/session1/university-1.md (visited on Feb. 2, 2017).

[5] *Ontop code*. [Online]. Available: https://github.com/ontop/ontop (visited on Feb. 2, 2017).

[6] *Obdalib plugin debug*. [Online]. Available: https://github.com/ontop/ontop/wiki/ObdalibPluginDebug (visited on Feb. 2, 2017).

[7] *Owl api examples*. [Online]. Available: https://github.com/ontop/ontop-api-examples (visited on Feb. 2, 2017).