Keeping modular and platform-independent software up-to-date: benefits from the Semantic Web

Olivier Dameron
Stanford Medical Informatics, Stanford University, USA
dameron@smi.stanford.edu

1 Introduction

Context We consider the problem of automating the maintenance of a platform-independent software environment composed of one main application, several plugins and some external third-party applications. We assume that the architecture is distributed and loosely coupled: each of these software component can be maintained by different groups and hosted separately, and no common release format is required.

Particularly, we consider the Protégé[1] ontology editor and knowledge base manipulation framework. In addition to the core system of Protégé, the community developed and provides about one hundred plugins[2]. Protégé can also be used in conjunction with independent external applications such as reasoners (e.g. Racer[3]) or inference engines (e.g. Jess[4]).

Approach We assume that (1) keeping up-to-date a local installation of Protégé can be automated, and that (2) the approach can be generalized to any similar environment.

For each software item, we have to perform the following steps: find the version of the latest available build; compare it with the version of the installed build; if necessary, find the download URL for the latest version, and update the installed item; and finally apply local customization.

Problem Among the previous tasks, it turns out that *the most difficult one is to automatically find the version and the URL of the latest build of the software items*. In this article, we consider the different possibilities for automatically retrieving the version number of the latest release and the download URL of software items. We analyze the limitations of the syntactic approach for parsing HTML pages or XML-based documents. We then show that they are limitations of the approach itself, that uses syntactic tools for performing a semantic task. We eventually propose a lightweight semantic-based solution allowing each provider to describe its software and that do not require additional tools on the client side. Moreover, this solution supports semantic heterogeneity, as every provider is free to extend the description according to their specific requirements without compromising the general compatibility.

2 Syntactic parsing of available HTML pages

For any human user, retrieving the version and downloading the latest release of each software item is straightforward from their respective web site. However, for a machine, the relevant information

[1] http://protege.stanford.edu
[2] http://protege.stanford.edu/download/plugins.html
[3] http://www.cs.concordia.ca/~haarslev/racer/
[4] http://herzberg.ca.sandia.gov/jess/

is buried into the HTML code, with no proper delimiter , and nothing to distinguish it from the noise.

Therefore, retrieving the relevant information requires some potentially complicated string manipulations with regular expressions based on the position of keywords. It is possible to devise some trick based on inserting keywords in comments in the right places. Moreover, this approach lacks robustness, for example if the developper changes the HTML code, invalidating our assumptions on the position of the information we are looking for.

We are confronted to an inner limitation of HTML: HTML is for humans, not for (smart) applications.

3 Syntactic parsing of XML or RDF documents

Using XML, it is possible to delimit the various bits of information, and to represent explicitly their structure. We could then ask all the contributors of the Protégé community to provide an XML-based description of their software.

However, there is no commonly accepted DTD nor XML Schema available. We could provide one, or maybe reuse the Eclipse[5] one, but it would still require independent applications such as Racer to adopt our own solution. Moreover, and this is the main limitation, XML is well adapted as long as all the participants have a shared understanding the data structure, but it does not support extension nor automatic reasoning about this structure. As a consequence, this approach lacks scalability. If some of the participants need to extend the common format, then the others would not be able to exploit the additional information.

RDF and RDF Schemas would allow to specify some machine-processable basic semantics, thus addressing the extensibility and heterogeneity issues of XML. Particularly, the DOAP project[6] provides an XML vocabulary to describe (open source) software projects. DOAP descriptions are provided for the ProtegeScript[7] and the Prompt[8] plugins.

The version and the download URL can be easily retrieved from a project's DOAP description. Moreover, a DOAP description refers to a DOAP RDF Schema[9] that provides an explicit and machine processable description of its semantics.

However, directly using DOAP has the following limitations. First, DOAP needs to be extended for representing various distributions of a single project, which is the case for the core Protégé: Protégé is available in a stable and in a beta versions, and the download URL also depends on the architecture and the inclusion of a Java Virtual Machine. Second, the DOAP description is still parsed syntactically. This is a problem with the XML syntax of RDF, as there exists several valid representations of the same statement.

4 Semantic querying of RDF documents

We addressed the previous two limitations.

We extended the DOAP Schema for representing Protégé-specific information[10] (show snapshot of the RDFS file in Protégé). The extended DOAP file[11] for Protégé is available online.

Using RDF queries over the DOAP descriptions allows the user to perform high level semantic queries such as "find the revision of the stable branch" in a SQL-like form independent of the RDF syntax variability.

[5] `http://www.eclipse.org`
[6] `http://usefulinc.com/doap`
[7] `http://smi.stanford.edu/people/dameron/protegeScript/doap.rdf`
[8] `http://protege.stanford.edu/plugins/prompt/doap.rdf`
[9] `http://usefulinc.com/ns/doap#`
[10] `http://smi.stanford.edu/people/dameron/ontology/rdf/doap-od.rdf`
[11] `http://protege.stanford.edu/doap/doap.rdf`

Moreover, RDF queries transparently handle the Protégé-specific specializations, so the previous query also works with the extended DOAP description of Protégé for example. However, requiring every client that would want to perform automatic update of its Protégé environment to have a RDF query engine locally installed seems impractical. Therefore, we propose to use an Ontology Web Service [1] providing a generic RDF-processing functionality[12]. The client only has to implement basic web service capabilities for sending the DOAP document and a SeRQL query to the server, and retrieving the corresponding version number and download URL.

5 Implementation of a solution

In order to demonstrate the feasibility of our approach, we implemented checkProtege[13], a Python script that automatically keeps a local installation of Protégé and of the related software up-to-date.

The checkProtege script implements the three approaches that we have examined. For some items such as Racer or the OWL-plugin for Protégé, for which there is no DOAP description available, it relies on syntactic parsing of HTML pages. For items having a DOAP description, such as Protégé, the Prompt plugin and the scripting console, the client can choose to perform a syntactic parsing of the RDF documents, or, more elegantly, to call the SeRQL engine web service.

Interestingly, this approach can be used to keep automatically checkProtege itself up-to-date before processing the Protégé-related items. When necessary, new functionalities are added, updating checkProtege automatically updates the configuration file too.

So far, checkProtege has been used successfully for four months on Windows, Apple and Linux machines, including one server for which the script is called automatically every night. The integration of checkProtege into Protégé itself is in progress. Special attention will be paid in separating a generic library of DOAP manipulation functions from the Protégé-specific parts.

6 Conclusion

We have presented a solution for the automatic maintenance of a composite software environment from distributed sources. Particularly, we showed that the problem of retrieving the latest available version number and download URL from each of the components without enforcing each of the contributors to provide a description in a common and fixed format is a semantic one. Consequently, we proposed a RDF-based solution and demonstrated that it could be extended by some of the participants without impairing the general compatibility. Eventually, we assessed to feasibility by implementing a demonstrator for the Protégé project.

References

[1] O. Dameron, N. F. Noy, H. Knublauch, and M. A. Musen. Accessing and manipulating ontologies using web services. In *Proceeding of the Third International Semantic Web Conference (ISWC2004), Semantic Web Services workshop*, 2004.

[12] `http://smi-protege.stanford.edu:8080/axis/services/rdfQuery`
[13] `http://smi.stanford.edu/people/dameron/script/checkProtegePython/`