Using the JOT plugin for reasoning with Protégé

Olivier Dameron

June 5, 2004

Abstract

JOT is a plugin that adds Python scripting capabilities to Protégé. It provides a direct access to the core of the ontology through the API. It is compatible with both the fame-based approach and the OWL plugin.

JOT is meant to be a complement to computationally-efficient dedicated reasoning services.

First, we show that JOT can be used in conjunction with such services. Second, we then show that JOT can be used to detect and to fix inconsistency that are beyond the realm of Description Logics based classifiers. Eventually, we show that JOT can be used as a quick implementation of intermediate representations.

1 Context

Maintaining ontologies involves handling tedious or complicated tasks such as adding concepts or relationships that depend on each other, applying some modeling patterns or making sure that the ontology complies to these patterns.

The JOT[1] plugin adds Python scripting capabilities to Protégé in order to address these requirements.

2 Architecture

JOT's architecture relies on a Java implementation of the Python language. It also reuses an implementation of a command-line console. These two components are wrapped together as a Protégé plugin.

Python commands can be executed directly by the user. They can also be saved in script files for future reuse. Eventually, thanks to the Java implementation, any compiled Java code can be called from Python script.

JOT communicates with Protégé by calling directly its Java API. The user is provided with a `kb` variable that represents the current knowledge base. From there, it is possible to reach the concepts and relationships.

JOT is compatible with plain Protégé ontologies as well as with the OWL plugin. In the first case, the `kb` variable is an instance of `KnowledgeBase`. In the latter, it is an instance of `OWLKnowledgeBase`.

[1] `http://smi-web.stanford.edu/people/dameron/jot/index.html`

3 Calling external reasoners

Because JOT has access to the Protégé API, it can use it to call an external classifier such as Racer if the ontology is an OWL ontology.

In addition, the JOT capability to execute arbitrary compiled Java code can be used to call any domain-specific library. Such libraries can provide domain-specific reasoning functions. Particularly, this principle can be used to access remote functionalities provided as Web Services.

4 Detecting and fixing semantic inconsistencies

JOT can be used to detect and to fix contradictions with the general knowledge of a domain that are beyond detection of insatisfiability by Description Logics classifiers. Such problems typically arise when two sets of concepts have the same structure or when there are dependencies between relationships.

The `Heart` has a structural part `WallOfHeart`, which is in turn composed of three structural layers: the `Epicardium`, the `Myocardium` and the `Endocardium`.

The `Heart` is also classically decomposed into four regional parts: the left and right atria and ventricles.

Therefore, we have to generate concepts such as `WallOfLeftAtrium`, which is both a structural part of `LeftAtrium`, and a regional part of `WallOfHeart`. In addition, we also have to deal with concept such as `MyocardiumOfRightVentricle`, which is a structural layer of `WallOfRightVentricle`, and a regional part of `Myocardium`.

In this example, the structural decomposition of an anatomical structures is propagated to its regional parts. Therefore, it is necessary to ensure that this pattern is respected. One solution is to generate automatically all the concepts and relationships. The other solution is JOT provides a convenient framework for implementing these two solutions.

Another example is the propagation of continuity relationships from the parts to the superstructure. In brain cortex anatomy, the anterior and the posterior parts of the subcentral gyrus are continuous. The former is a part of the frontal lobe, whereas the latter is a part of the parietal lobe. From the original continuity relationships, we can infer three other continuity relationships, e.g. between the anterior part of the subcentral gyrus and the parietal lobe, or between the fronatl and the parietal lobe.

Representing such dependencies relationships can be represented by implications that require the use of variables. No OWL-compatible implementation of such a language exists yet. In the meanwhile, such implications can be easily expressed in Python either for checking that they are respected, or for actually generating the dependent relationships.

5 Intermediate representations

Intermediate representations provide a consise and high-level view on a domain. They allow to hide the portions of an ontology that are nor relevant to a partic-

uliar domain, as well as some implementation details. Not only do they require extended expressivity, but also extended reasoning capabilities. JOT can be seen as a tool for creating such intermediate representations, and for using them.

We took advantage of Python's capability to define functions and to save them into separate files. A typical usage of the JOT plugin consists in loading such external function, and then in a succession of fonction calls providing higher-level functionalities. These calls can then be seen as a representation of the model in a concise intermediate representation. The mapping between this representation to the actual formalism, be it OWL or the classical frame-based Protégé.

So far, we have identified two kinds of functions.

The functions of the first one implement some design pattern. They are dependent of the representation language, but are domain-independent. For instance, in OWL such a function makes a concept equivalent to the union of its direct subclasses, and makes such subclasses disjoint. It can be used to easily provide an enumeration of the fingers (i.e. a `Finger` has to be exactly one of `Thumb`, `Index`,...), as well as an enumeration of `Wine` as `RedWine`, `WhiteWine` or `RoséWine`. In the frame-based Protégéformalism, it would require slightly different steps.

The functions of the second set are domain dependent, but independent of the representation formalism. Typically, they reuse functions from the previous set. In the domain of anatomy, such functions allow to create lateralized concepts (i.e. concepts that are either on the left side or on the right side of the body, such as `Thumb` or `Lung`). Here, it consists of creating the concept and its disjoint left and right subconcepts, and then in stating that the concept is equivalent to the union of its subconcepts, which can be done by the function detailed for the first set. Another example is the function allowing to create an enumerated anatomical concept (e.g. `Vertebra` and its thirty three subconcepts), or even an enumerated lateralized anatomical concept (e.g. `Rib` and its twelve subconcepts, each in turn subsumed by a left and a right subconcept).

6 Conclusion

JOT can be used as a complement to dedicated reasoning tools for interacting with ontologies in Protégé. It can be used for ontology-maintenance tasks requiring *ad-hoc* primitives. Such a feature can also be exploited for providing intermediate representations of ontologies.