Representing and Using Template-Knowledge for a Medical Ontology in Protégé

Ronald Cornet
dept. of Medical Informatics
AMC - Universiteit van Amsterdam

Michel Klein
dept. of Artificial Intelligence
Vrije Universiteit Amsterdam

Abstract

We present a medical ontology that is used for registering health problems at intensive care units in hospitals. Because of its flexible architecture it is necessary to be able to determine equivalence and subsumption automatically. These two tasks require two types of knowledge to be encoded: definitional knowledge and template knowledge. We discuss how both kinds of knowledge can be made explicit and used in Protégé, using the OWL language. In our approach, we have to mix description logic (DL) style of knowledge representation with a frame-based style of representation. The consistency between both is an important issue. Our aim with this paper is to demonstrate and discuss approaches to augment DL-based representations with frame-based meta-knowledge.

1 Real-World Usage of Ontologies

DICE (Diagnoses for Intensive Care Evaluation) is an ontology regarding reasons for admission in intensive care. Its primary applications are providing a means for registration and aggregation of patients' reasons for admission. In order to stimulate clinicians and nurses to provide as much detail as is known about a patient, a mechanism for supporting post-coordination has been implemented, based on a (frame-based) representation that allows specification of refinable characteristics. Aggregation is supported by means of definitional characteristics, that enable grouping together reasons for admission that share common properties.

As the post-coordination mechanism allows for a multitude of ways to specify a reason for admission, it is necessary to be able to determine equivalence and subsumption automatically. In order to realize this, a DL-based representation is being investigated. Ideally, this will contribute to the development of an ontology that is consistent, supporting highly detailed registration, allowing automated classification and instance querying.

As an example of the post-coordination mechanism, consider a patient being admitted to the intensive care department with acute type B viral hepatitis. The mechanism provides various ways of constructing this expression, e.g. as Viral Hepatitis, *caused by* Hepatitis B virus, *having course* Acute, as Acute Viral Hepatitis, *caused by* Hepatitis B virus, or even as Infection, *located in* Liver, *caused by* Hepatitis B virus, *having course* Acute.

2 Description Logic versus Frame-based Knowledge

The two tasks of automated classification and registering instance information mentioned in Section 1 require two types of knowledge about a concept to be described: knowledge about its semantics, i.e. its *definitional knowledge*, and knowledge about the construction of instance data, which we call *template knowledge*. These two issues are related, but certainly not interchangeable. For example, consider the concept 'wine'. 'Wine' can be defined as "the fermented juice of fresh grapes used as a beverage" (Merriam-Webster Online Dictionary). For knowledge acquisition purposes however, one wants to specify that the name, color, sugar, flavor, grape, maker, and body of the wine are relevant properties of wine [2]. Red wine can be defined as "wine which has a red color", and for red wine the tannin level can be specified.

In the ontology introduced above, we use the description logic (DL) based characteristics of OWL to describe the semantics of concepts. The benefit DL is the possibility of reasoning with defined necessary (and sufficient) properties of concepts. At the same time, we used frame-based representations to represent template knowledge. Frames offer

the possibility of specifying relevant properties of concepts. However, when both are required, there is the need to guard consistency between relevant (frame-based) properties and necessary (DL-based) properties.

3 Representing Template Knowledge

The template knowledge that is relevant in our ontology are the post-coordination rules. These can be seen as specifications of how subsumees of specific concepts in the ontology can be created by combining existing concepts. Concepts that allow for post-coordination have a number of properties for which values are defined (i.e. the definitional knowledge), and others for which 'choices' are defined (the 'template knowledge'). For example, `Viral_Hepatitis` is described as follows:

```
Viral_Hepatitis
  def: has_location: Liver
  def: system_involved: Digestive_System
  def: has_aetiology: Virus
  templ: has_aetiology: >= 1 children-of
     { Hepatitis viruses,
       Epstein-Barr virus,
       Cytomegalo virus }
```

The lines following '**templ**' describe that instances and subsumees of `Viral_Hepatitis` can be defined using one or more children from the listed concepts.

There are several ways in which such knowledge can be represented in Protégé/OWL. Each of them has a number of disadvantages. We discuss three approaches below.

3.1 Using ∀-restrictions

The first approach is to represent every template property via ∀-restrictions in the specifications of the classes themselves, i.e. using the `owl:allValuesFrom` construct. This is logically correct because all subsumees and instances should choose their values from the classes specified in the template property. However, conceptually a template property is different from definitional knowledge. For example, consider the description of `Viral_Hepatitis` in the previous section, where `has_aetiology` has a role both in the definition and as template property. Using this approach, template properties are undistinguishable from definitional properties and different types of knowledge are represented in the same way. Moreover, additional knowledge about the number of fillers for a template property cannot be represented.

3.2 Domain of Properties

Another approach is to specify the union of all classes on which a template property can be applied as domain of the property. The major drawbacks of this approach are that it provides a global specification instead of local (at class level). As a result, it is not possible to specify different ranges for different classes. This can be overcome for example by defining class-specific subproperties (e.g. viral-hepatitis-cause, having Viral Hepatitis as domain and Virus as range), but this will lead to large numbers of properties.

3.3 Using Meta-Classes

A third approach is the use of meta-classes with additional property restrictions. In this approach, we define a meta-class `RefinableClass` as subclass of `owl:Class` and a meta-class `Template` with three properties: `refinableProperty`, `valueType` and `multiplicity`. In the OWL abstract syntax the definition is as follows:

```
Class(RefinableClass partial owl:Class)
ObjectProperty(hasTemplate
        domain(RefinableClass)
        range(Template))
Class(Template partial
```

```
restriction(refinableProperty allValuesFrom(rdf:Property))
restriction(valueType allValuesFrom(owl:Class))
restriction(multiplicity allValuesFrom(xsd:int))))
```

Classes for which we want to represent template knowledge are made instances of `RefinableClass` instead of `owl:Class`. For these classes we can specify templates with actual values for the properties `refinableProperty`, `valueType` and `multiplicity`, in addition to the general class axioms (the definitional knowledge).

This approach has two disadvantages. First, by using meta-classes we do not stay inside the OWL DL sublanguage, but we create an OWL Full ontology. As a result, DL reasoning cannot be done anymore on the ontology as a whole.

A second problem is the possible discrepancy between the definitional knowledge and the template knowledge. For example, nothing in this approach prevents modelers to specify value for the `valueType` property that are inconsistent with the values defined in the restrictions on the class itself.

4 Representing DICE in Protégé

For representing DICE in Protégé we investigate the meta-class approach. The reasons for this choice are the fact that it allows for the most complete specification of template knowledge and that there are additional measures possible to solve the disadvantages. Moreover, we think that it is conceptually the cleanest approach of the three options.

Knowledge about refinable classes can be interpreted procedurally (functionally) as a "request for further specification". Ideally, a knowledge-acquisition environment should present the allowed property values to a user, and provide possibility to specify one or more values. In order to achieve this in Protégé, we will have to implement an extension that interprets the `RefinableClass` specification.

4.1 Solutions to Problems

To use the meta-class approach, we will have to overcome the problems mentioned above. With respect to the reasoning problem, we could split the ontology into two separate OWL-files: one with the definitional knowledge only, and one with the meta-class and the actual values for the meta-class properties for the refinable classes. The first file will be used for reasoning tasks, the combination of the two files for other tasks. Maintaining and editing the ontology in Protégé will be problematic in this scenario, as it is not possible to specify in an editor how ontology axioms should be distributed over the different files (at least up to Protégé version 3.0). A second way to cope with the reasoning problem is to filter out the statements that refer to the meta-class and its properties before performing a reasoning tasks.

We plan to solve the problem of a possible inconsistency between the definitional (DL-based) knowledge and the template (frame-based) knowledge by specifying 'invariants' for the ontology that always should hold. The invariants specify a relation between different aspects of the two types of knowledge that should be maintained, preferably by the editing environment.

4.2 Invariants

The following invariants should hold for ontologies that use the meta-class as described in Section 3.3 to specify template knowledge.

Only refining refinableProperties Each subclass of an instance of a `RefinableClass` should only define property restrictions for properties specified as value of the `refinableProperty`. In other words, if a subclass specifies a filler for a property, this implies this is a `refinableProperty` for the superclass, hence it should be defined as such. In our example, subclasses of `Viral_Hepatitis` are only allowed to define property restrictions for the property `has_aetiology`.

Adhering to valueTypes for refinings Any specified property value for a subclass of an instance of a `RefinableClass` must be a logical subset of the value of the `valueType` property in the template. In our example, subclasses of `Viral_Hepatitis` should only use values for property restrictions on `has_aetiology` that are a subset of the union of the `Hepatitis viruses`, `Epstein-Barr virus`, and `Cytomegalo virus`.

Inheriting valueTypes Similarly, refinable subclasses of an instance of a `RefinableClass` should only use fillers of the `valueType` that are a subset of the fillers of the `valueType` on the same property of all of its superclasses.

Consistency between valueType and allValuesFrom The value of the `valueType` property in a template should always be a logical subset of the value of an `allValuesFrom`-restriction on the same property within the class. This situation does not occur in our example, but if there would have been an `allValuesFrom`-restriction on `has_aetiology`, its value should have been a superset of the union of `Hepatitis viruses`, `Epstein-Barr virus` and `Cytomegalo virus`.

Template absorbtion Each subclass of an instance of a `RefinableClass` inherits the templates of its superclasses. However, when a subclass specifies a property restriction for the property in the `refinableProperty`, there should be no template for that property for the subclasses of that class.

For the implementation of the invariants, we investigate several options. There are two levels of implementation: detection of the violations and enforcement of the invariants.

The OWL-plugin of Protégé allows for the implementation of *ontology tests*. Ontology tests are pieces of Java-code that are executed when Protégé is asked to check the model. Given the fact that the complete knowledge model is accessible from the Java-code, we expect that all violations of above invariants are detectable with this implementation.

Using PAL constraints is another option for detecting violations. Using them, we can use the existing Protégé-machinery for detecting and enforcing constraints. It remains to be determined whether all invariants can be encoded in PAL.

Finally, we think about using rules, e.g. specified in SWRL [1]. We are not aware of any automated support for checking or enforcing SWRL rules in OWL Full ontologies. It is also unclear whether SWRL rules are expressive enough to encode all invariants.

5 Discussion

Although in this article we discuss the use of template knowledge for encoding post-coordination rules, the issue is more general. When constructing ontologies, people often have ideas about how to structure the ontology. Sometimes, this knowledge is made explicit, e.g. via written guidelines, sometimes it is only in the mind of the developer. Ideally, this 'meta-knowledge' is encoded in the ontology itself, because it remains important during the whole lifetime of the ontology: changes to the ontology should not violate the structural principles. This knowledge could be encoded as template knowledge as well.

Besides the post-coordination rules, one could think of general or domain-specific structuring guidelines that aim at a good maintainability, e.g. the ones listed in [3]. An example of a general guideline is "the branches of the primitive skeleton of the domain taxonomy should form trees". Combining this meta-knowledge with an OWL-based specification of ontologies requires mixing frame knowledge with description logic knowledge. The consistency between both types of knowledge is not automatically guaranteed. Specific 'invariants' will have to be defined for these types of knowledge.

Acknowledgements

This research was supported by the Netherlands Organisation for Scientific Research (NWO) under project number 634.000.020. See `http://www.i-catcher.org/`.

References

[1] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A semantic web rule language combining OWL and RuleML. W3c member submission, World Wide Web Concortium, May 21, 2004.

[2] N. F. Noy and D. L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report SMI-2001-0880, Stanford Medical Informatics, Mar. 2001.

[3] A. Rector. Modularisation of domain ontlogies implemented in description logics and related formalisms including OWL. In *Knowledge Capture*, pages 121–128, Sanibel Island, FL, 2003. ACM.