Differences between UML and Protégé-2000 Knowledge Models

Caveats:

This document is really just a first guess at the sort of semantic mismatches between UML and Protégé-2000. To build an import/export for UML (via XMI), I am assuming that all of the below issues will have to be resolved in one way or another. In many cases (especially methods and instances), we must simply assume that the import/export facility will lose information. In other cases, the appropriate choice is not as clear.

For some of these mismatches, it could be that I am misunderstanding or missing some important information about UML and XMI. I haven’t yet looked into any of the syntactic details, and it could be that something I now think will be a problem won’t. However, by the same reasoning, something I think will be easy to resolve, actually could be a real show-stopper.

In general, I am much more familiar with Protégé than with UML, so I list things one can say in Protégé and not UML. There may be things one can say with UML that I have omitted. I welcome feedback about omissions or any other comments folk may have on this brief document (send to gennari@u.washington.edu).

Methods & Instances:

First, there is the huge distinction that UML allow for methods to be associated with classes, and to describe the input/output signature of those methods. Protégé is only about classes, attributes (slots) and the various relationships among classes—it has no capability to store or reason about methods. Thus, any import/export facility for Protégé and UML will certainly have to drop all method information.

Conversely, Protégé knowledge bases include both classes (which map nicely to UML classes) and instances. The latter sort of object does not usually map to anything in UML (but see discussion of meta-classes, below). Thus, I would assume that an export to UML from Protégé would necessarily throw away any instances (and any references to instances) in the knowledge base.

However, even if UML methods and Protégé instances are completely ignored, there is a lot of information in the class organization and the list of attributes which could be imported and exported to and from Protégé. Unfortunately, even with this omission, there are a large number of knowledge-level differences that will make it difficult to go back and forth between UML and Protégé. Each of the following issues will have to be resolved one way or another:

Attributes vs Relations

In UML, a distinction is made between atomic attributes (e.g., single-cardinality attributes of type integer, string, etc.) and relations which may be labeled and connect a class to some other class. In Protégé, both sorts of objects become slots. One challenge is that in many UML dia-

---

1 There has been a fair amount of work within the Protégé framework to capture what is known as “problem-solving methods”, and it may be that this work can be leveraged for UML method signatures. However, I will certainly leave this to others. (Among others, Monica Crubezy at Stanford has worked in this arena.)
grams, relations are not named, so we will need to generate Protégé slot names for these. But which class should get the relation slot? One possibility is to give it to both classes, and enforce an inverse relationship among these two slots. Going from Protégé (non-atomic) slots to UML relations will also be challenging.

**Slots as first-class objects**

In Protégé, slots are first-class objects, with most of the same attributes and behaviors as classes or instances. Most importantly, slots are not defined as local to a particular class: In a given knowledge base, all slot-names must be unique. One can, of course attach or use a particular slot (like “name”) in multiple classes, but only if the semantics of that slot are identical.

In UML, one is allowed to have two slots with the same name, but (perhaps) different data types in two different classes. This is not allowed in Protégé, and such slots would have to be renamed (by pre-pending the class name, perhaps).

**Inheritance:**

In UML, classes can be related to each other via *generalization*. This must be mapped to the Protégé notion of an is-a hierarchy and inheritance. Both allow systems allow for attributes to be inherited from multiple parents. One difficulty is that in Protégé, all template slots (including relations) are inherited, whereas I think that UML only inherits atomic attributes, and not relations.

**Meta-classes**

In Protégé, an important capability (even if not used by most users) is provided by the notion of *meta-classes*. This is complex to explain briefly, but affects inheritance, and leads to the distinction between *own-slots* and *template-slots*. Because UML does not explicitly deal with any instances, I think that the notion of meta-classes will largely have to be ignored when exporting Protégé KBs to UML.

**Packages:**

In UML, one often groups together a whole set of classes as a *package*. This becomes an abstraction for the set of classes (and methods) and can in turn be used as an element in even higher-level packages.

The closest feature to this idea within Protégé is the inclusion mechanism. However, inclusion does not do any abstraction, and has different semantics than packages. In particular, when one is editing a project that has included ontologies, one is not allowed to edit any of the content from that ontology. These semantics improve the re-usability of the “base-level” ontologies, so that they can be included into several projects. This behavior is very different from UML packages, which are designed to support information-hiding & abstraction, rather than reuse.

In general, I suspect that it will be difficult / impossible to capture UML packages in Protégé. Since most large UML diagrams use packages, this could be a real handicap to the use of Protégé for UML diagramming.

**Composition, Aggregation, & Partonomies**

The notion of “part-of” is captured in UML via the aggregation or the composition relationship between classes. Protégé does not have any mechanism for treating part-of as a special relationship—it will have to be simply another slot. This means that when exporting from Protégé, it will be very difficult (impossible) to reconstruct aggregation or composition relationships in UML.
Data types
As might be expected, there are some data type mismatches between UML and Protégé. Mostly UML describes objects above the level of data types, since it has no instances. For example, one can specify attributes w/o any data type in UML. These will have to be given a default Protégé type, presumably string.

Protégé allows for multiple-cardinality slots that would be UML attributes: E.g., a slot can have a set of integers, or a set of strings. I suspect that this would have to be ignored / dropped when exporting to UML.

Protégé also has data types for either “instance” or “class”. Attributes that have these datatypes will become UML relations, but the distinction between the two will have to be dropped.

Default values
Protégé allows one to give a default value for most slots. When instances are created, this value is filled-in as a default. Since UML does not deal with instances (or slot values), this information would have to be dropped when exporting to UML.