An approach to cross-language model versioning

Harald Störrle
Institut für Informatik, Universität Innsbruck

May 15, 2007

1 The UML-world is not enough

Using models is considered to be an industry best practice for a great many situations, including the early phases of large software development projects, schema integration for databases, and business process management and optimization. Very often, large families of models are created in such settings and many of the models may have a prolonged lifetime with numerous changes, additions, and updates. Thus, versioning of models is imperative for real life projects.

Today's approaches to model versioning are very restricted. They might start with the UML modeling language, consider only one model type such as static structure models (aka. "class models"), and build heavily on the UML meta model as the data structure and XMI as the file format. For instance, this scenario is used in the Fujaba-approach.[1] But there are many other scenarios as well. We will spend the next few paragraphs discussion shortcomings of the scenario mentioned. First, observer that there are (at least) three dimensions of models that are relevant for versioning:

modeling language the notation or family of notations in which a model is described (including, of course, textual as well as visual notations);

model type the family of concepts used for modeling together with their properties and relationships;

data structure the data structure in which a model is stored in memory, or the file format in which a model is stored.

Concerning the modeling language, although UML is claimed to be the "*lingua franca of software engineering*", there are still many other modeling languages in widespread use today. For instance,

- for embedded/real time systems modeling (e. g. in the telecom industry), SDL/MSCs are the most common modeling notation;
- for business process management and modeling (e. g. in the standard software and customizing businesses), ARIS/EPCs are the most common modeling notation;
- for corporate data models, database design, and ontology modeling, ER-like models are still more common as a modeling notation than UML.

And even when focusing on the UML as the modeling language for a difference algorithm, there are 13 different diagram types defined in the UML, all of which have several types of usage and specialisation constituting an individual model type each. And then, there are dozens of other modeling language families with their own special kinds of models.

And even when focusing on the UML as the modeling language and class diagrams as the type, there are different UML-version to chose from, different XMI-versions, a multitude of tools with their own interpretations of UML and XMI, their own bugs and patches and so on.

That is, even in the limited scenario we have mentioned at first, there are still many variation points, each of which may have a significant influence on a difference algorithm, resulting in false positives or false negatives when comparing models created with, for instance, different versions of the same tool. Only when all these parameters are fixed do we have a sufficiently precise foundation for the definition of a difference algorithm, and thus versioning.

That means, on the other hand, that almost all combinations of modeling languages, model types, and data structures are actually incompatible, even if there are conceptual similarities that should be found by a difference algorithm. To some degree, this problem may be circumvented by automated model transformations into, say, a certain combination of UML and XMI versions of a certain tool. This does not work for all cases, though, for instance, many text-based models such as the popular use case tables have not counterpart in UML. Similarly, ordering of use cases in use case maps is not part of the UML. The same problem arises even more stringently for Domain Specific Languages.

Another drawback is efficiency: XMI – as all XML formats – is dreadfully verbose. So, even small models yield large XMI files, and real life examples may easily

[1] See [KWN05]. Fujaba 4.3 accepts only class models conforming to UML 1.3 and XMI 1.2.

reach hundreds of megabytes or even several gigabytes in size.[2] Large files are inefficient to process for a variety of reasons, most notably, exhausted capacity of disc caches and main memory, resulting in e. g. page swapping.[3] Unlike multimedia data, models are not streams of data. Rather, they are tightly interwoven so that most operations cannot be implemented for pipeline-style processing, including differences. Thus, processing XMI comes with a hefty linear factor both for run time and memory footprint. This penalty does not deteriorate the algorithmic complexity, of course, but it predominates practical cases. And all of this just because of the inefficient model encoding of XMI.

Therefore, we propose a two-step approach to difference computation. In the first step, models from all sources are transformed into a common meta-metamodel for differencing. The encoding of models in this meta-metamodel is much more efficient than XMI. In a second step, the differences between models is computed with more or less standard algorithms. See Figure 1 for a visualisation of this approach.

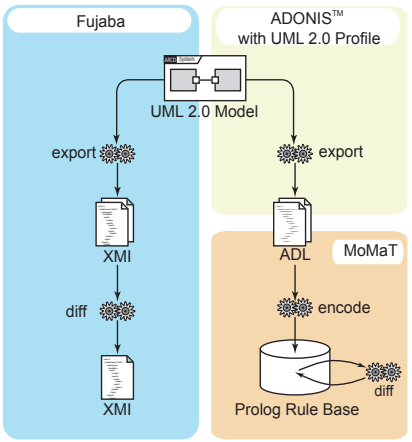

Figure 1: The Fujaba-approach (left) and our approach (right).

2 A common Meta-Metamodel for all modeling languages

In our approach, a model is first transformed into a efficient representation according to the Meta-Metamodel shown in Figure 2. It should be obvious, that it is possible to transform any metamodel into this simple data structure.

Technically, instances of this Meta-metamodel are encoded as sets of Datalog-clauses (cf. [CGT90]), that is, as simple logical predicates that could also be mapped into a traditional relational schema. In this transformation, each

[2]This is the reason why tools like MagicDraw compress model files by default.

[3]It is not uncommon, that up-to-date desktop computers are not capable of processing a large XMI-based model in-memory, let alone with the default JVM-settings.

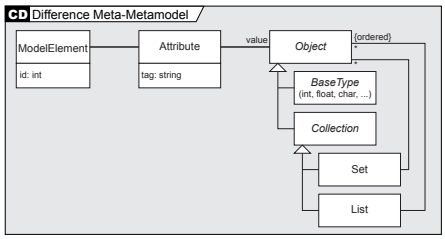

Figure 2: A Meta-metamodel as a normal form for arbitrary modeling languages.

model element becomes a Prolog-fact of the form `me(id, [tag-value, ...]).`, where `id` is an arbitrary unique identifier (typically an integer), `tag` is any atom to identify an attribute of the model element, and `value` is the value of this attribute. Consider Figure 3 for an example.

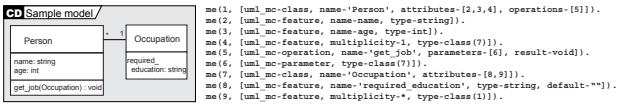

Figure 3: An example for the transformation from a UML class diagram (top) into a set of Datalog clauses (bottom).

The value of attributes may be arbitrary Prolog-terms. When the values of all tags in a model element are either atoms, lists of atoms, or terms `set(x)` where `x` is one of the former, the model element is said to be in normal form. When all model elements of a model are in normal form, the whole model is in normal form.

It is now easy to define predicates for computing the similarity of model elements, the difference of models, and so on. In contrast to implementations in, say, Java, the rules defining when two elements are supposed to be similar, and what should be considered a difference is straightforward due to the declarative nature of prolog programs.

References

[CGT90] Stefano Ceri, Georg Gottlob, and Letizia Tanca. *Logic Programming and Databases.* Surveys in Computer Science. Springer Verlag, 1990.

[KWN05] Udo Kelter, Jürgen Wehren, and Jörg Niere. A Generic Difference Algorithm for UML Models. In Klaus Pohl, editor, *Proc. Natl. Germ. Conf. Software-Engineering 2005 (SE'05)*, number P-64 in Lecture Notes in Informatics, pages 105–116. Gesellschaft für Informatik e.V. 2005.