Delta-P: Model Comparison Using Semantic Web Standards

Martín Soto
Fraunhofer Institute for Experimental Software Engineering
Kaiserslautern, Germany
soto@iese.fraunhofer.de

Abstract

The Delta-P [1] approach relies on Semantic Web notations and techniques to perform comparison of potentially complex models. The key feature of the approach is to allow for flexible adaptation, by using queries to detect changes relevant to different schemata. Although the approach was originally oriented to software process models, in this paper we concentrate om its potential applications to other types of models (e.g., UML). We briefly describe the approach and its initial implementation (Evolyzer) and discuss the direction of our future work.

1. Introduction

After more than 30 years of continuous progress, software configuration management offers us a wide selection of powerful and highly reliable software versioning tools. However, it strikes one as surprising that despite the advanced state of the discipline, and given the widespread use of modeling in modern software development, no complete solution is available that supports versioning and collaborative work on software models in an efficient and reliable way.

Indeed, a good portion of the theory that has been developed to support software versioning could also be applied to models. The main obstacle is probably the fact that most current implementations rely on line-oriented text comparison algorithms (like the well-known UNIX `diff` algorithm) for comparing and merging versioned objects. These algorithms work quite satisfactorily on source code text, but are pretty ineffective on files of other types. Particularly, the fact that comparison made at file level is normally very hard for human beings to understand and work with, represents a large limitation to the usefulness of standard versioning on software-related models.

This situation has motivated us to work on Delta-P, an approach for model comparison that processes models at a level of abstraction that is appropriate for humans to work with. In so doing, we have found a number of notations and techniques originally developed for the Semantic Web to be especially useful for the task. In this paper, we briefly describe our model comparison approach. This approach was originally conceived in the context of software process modeling, but we consider it general enough to be adequate for comparing models of other types. Right now, we are taking the first steps to extend it to generic UML models, and expect to have a demonstration available at the time of this workshop.

The rest of the paper is organized as follows: Section 2 discusses the model comparison problem in general, and provides rationale for our approach. Section 3 describes the approach in more detail and presents some examples of its use, while Section 4 briefly discusses the current implementation. Finally, Sections 5 and 6 close the paper by presenting some relevant related research, and discussing our options for future work.

2. The Model Comparison Problem

As mentioned above, line-oriented algorithms such as `diff` are generally not appropriate for comparing models. This does not mean, however, that the general approach used by `diff` is completely inadequate for model comparison. The literature on graph similarity and pattern recognition offers many relevant techniques (see [2] for an introduction) that rely on structural similarities to produce a mapping between corresponding elements of two model versions. These algorithms seem to have two characteristics in common: First they are generally very demanding of computational power and, second, they are not always 100% reliable, often requiring human intervention to ensure the quality of the final results. We acknowledge that these approaches have a large number of useful applications, but deem them impractical for day-to-day use in comparison and versioning.

Fortunately, in many practical cases, an element mapping is already available through the fact that modeling tools mark model elements with unique identifiers. The XMI standard, for example, specifies the `id` and `guid` attributes to store unique entity identifiers. `Id` is mandatory, and although tools are not forced to keep `id` values stable between work sessions, most of them actually do it. Also, our experience shows that modeling tools in other areas (the motivation for the author's work comes from the software process modeling domain) also often provide reliable unique identifiers for model elements.

By using unique identifiers, it is computationally inexpensive ($O(n \cdot \log(n))$ in worst case) to reliably map the corresponding model entities between two

model versions. This makes it possible to efficiently identify specific changes (e.g., *the cardinality of a relation was changed*) as well as to find entities that, for example, were deleted or added from one version to the next. The previous fact would suggest that the model comparison problem is, at least for many practical applications, a trivial one. We claim, however, that the basic comparison that results from matching identifiers between two model versions is just the first step of a comprehensive comparison process, one that offers a wide variety of useful possibilities to both comparison implementors and end users.

One of the major problems when comparing models is to properly identify and characterize all possible types of changes that may occur when using a particular schema. Figure 1 contains a (not necessarily exhaustive) list of changes that may occur to a UML class diagram. One important observation is that this list is quite heterogeneous, including changes that affect various sorts of elements in different ways. In the case of UML class diagrams, it is probably sensible to write software that considers each one of these cases explicitly. On the other hand, this could result too onerous for more specialized, less common schemata, or for schemata that change more often.

Class added
Class deleted
Class name changed
Attribute added to class
Attribute removed from class
Attribute type changed
Attribute moved from one class to another
Method added to class
Method removed from class
Method signature changed (can be made more detailed, i.e., return type changed, parameter added, etc.)
Relation added
Relation deleted
End point of relation changed
Relation name changed
Relation cardinalities changed (only for some types relations)
Relation type changed (e.g., from reference to aggregation)

Figure 1: Possible change types in a UML class diagram.

We propose to deal with this "change heterogeneity" by expressing the models first in a highly normalized representation. In this representation, a wide range of changes can be described uniformly by specifying change types formally as patterns. It is afterwards possible to recognize changes of a given type by recognizing their corresponding patterns in the compared models.

3. The Delta-P Model Comparison Approach

By making use of notations and techniques drawn from the Semantic Web, we have developed a comparison approach that achieves the following goals:

- Operate on a variety of schemata. New schemata can be supported with relatively little effort.
- Be flexible about the changes that are recognized and how they are displayed.
- Allow for easily specifying change types that are specific to a particular schema or even to a particular application.
- Be tolerant to schema evolution, by allowing the comparison of model instances that correspond to different versions of a schema (this sort of comparison requires additional effort, though.)

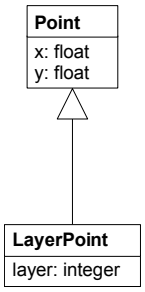
3.1. General Approach

More specifically, our approach can be decomposed in the following steps:

1. Convert the model instances to a normalized triple-based notation. We use the RDF notation for this purpose.
2. Perform an identity-based comparison of the the resulting RDF models, to produce a so-called *comparison model.* This model corresponds roughly to the results of a simple comparison based on unique identifiers.
3. Find relevant changes by recognizing the corresponding change patterns in the comparison model.

3.2. The Resource Description Framework

The Resource Description Framework (RDF) [3] was originally defined as a generic notation for describing resources in the World Wide Web. In actuality, however, it can be seen as a highly generic notation for representing arbitrary sets of objects and their relationships.




```
(c1, type, Class)
(c1, name, "Point")
(a1, type, Attribute)
(a1, name, "x")
(a1, dataType, "float")
(a2, type, Attribute)
(a2, name, "y")
(a2, dataType, "float")
(c1, hasAttribute, a1)
(c1, hasAttribute, a2)
(c2, type, Class)
(c2, name, "ProjectedPoint")
(a3, type, Attribute)
(a3, name, "layer")
(a3, dataType, "integer")
(c2, hasAttribute, a3)
(c2, inheritsFrom, c1)
```

Figure 2: A small UML class diagram and a possible RDF representation for it.

RDF models are sets of so-called *statements,* each of which makes an assertion about the world. Statements are always composed of three parts, called the *subject*, the *predicate*, and the *object*. The subject is a unique identifier of an entity. The predicate identifies an attribute of this entity or a relation the entity has with another entity. Depending on the predicate, the object contains either the value of the attribute or the identifier of the entity pointed to by the relation. For example, the very small UML model at the left side of

Figure 2 could be represented in RDF as shown on the right side of the figure. Although a relatively high number of statements is necessary to cover all details, the final representation is highly uniform.

Our choice of RDF as the base notation for comparison can be controversial. It could be argued, for example, that the Meta Object Facility (MOF) would be a more appropriate choice, especially when dealing with UML models. Our reasons to choose RDF can be summarized as follows:

1. It is generally inexpensive and straightforward to convert models to the notation.
2. Models do not lose their "personality" when moved to the notation. Once converted, model elements are still easy for human beings to recognize.
3. The results of a basic, unique-identifier-based comparison, can be expressed in the same notation. That is, comparisons are models too. Additionally, elements remain easy for human beings to identify even inside the comparison.
4. We already have a standardized notation (SPARQL) to express patterns in RDF graphs. With minimal adaptations, this notation is adequate to specify interesting types of changes in a generic way. Our Evolyzer system (see Section 4) provides an efficient implementation of SPARQL that is adequate for this purpose.

3.3. The Comparison Model

Since RDF statements refer to the modeled entities by using unique identifiers, two versions of a model expressed as RDF can be efficiently compared to determine which statements are common to both versions, and which are exclusive to only one of them. For the comparison of two versions A and B, we express the results of this comparison by constructing the union set of A and B, and *marking* the statements in it as belonging to one of three subsets: statements *only* in A, statements *only* in B, and statements common to A and B. The resulting set, with the statements marked this way, is called the *comparison model* of A and B.

One central aspect of the comparison model is that it is also a valid RDF model. The theoretical device that makes this possible is called *RDF reification,* and is defined formally in the RDF specification. The main purpose of RDF reification is to allow for statements to speak about other statements. This way, it is possible to add assertions about the original model statements, telling which one of the groups above they belong to.

3.4. Identification of Specific Changes

In most practical situations, comparison models are too large and detailed to be useful for direct human analysis. The main problem lies in the fact that most changes that human beings perceive as a unit correspond potentially to many added and/or deleted statements. For example, the change "class `c1`'s name was changed from 'Poitn' to 'Point'" corresponds in RDF to removing the statement `(c1, name, "Poitn")` and adding the statement `(c1, name, "Point")`. More complex changes such as adding a new class may involve a much larger number of statements.

For this reason, a mechanism that makes it possible to automatically identify changes at a higher level of abstraction is necessary. In our approach, this mechanism is provided by the SPARQL RDF query language [4]. SPARQL queries are made of a number of patterns that are intended to be matched to the statements in a model. When a pattern matches a statement, variables in the pattern are instantiated with the corresponding values from the matching statement. A result is returned for every possible combination of statements from the model that matches all patterns in the query.

Although a detailed description of SPARQL is out of the scope of this paper, Figure 3 shows three queries that are capable of identifying basic changes in a UML class diagram comparison (which uses a schema similar to that illustrated in Figure 2.) The first query identifies added classes, the second finds renamed classes and returns their old and new names, and the third identifies classes with new attributes and displays their names. This examples can give the reader an idea of the level of concision with which changes can be described.

```
select ?class
where {
  graph comp:B {?class rdf:type uml:Class} }

select ?class ?oldName ?newName
where {
  graph comp:AB {?class rdf:type uml:Class}
  graph comp:A {?class uml:name ?oldName}
  graph comp:B {?class uml:name ?newName} }

select ?class ?attr ?attrName
where {
  graph comp:AB {?class rdf:type uml:Class}
  graph comp:B {?attr rdf:type
                  uml:Attribute .
                ?attr uml:name ?attrName} }
```

Figure 3: SPARQL queries identifying some basic change types in UML class diagrams.

4. Implementation

Our current implementation of model comparison was especially designed to work on large software process models, such as the German V-Modell [5] and its variants. Nevertheless, since the comparison kernel implements a significant portion of the RDF specification (with the remaining parts also planned) support for other types of models can be added with relatively small effort.

The current implementation is written completely in the Python programming language and uses the MySQL database management system to store models. Until now, we have mainly tested it with various process models, including many versions of the V-Modell (both standard releases and customized versions.) Converted to RDF, the latest released version

of the V-Modell (1.2) contains over 13.000 statements that describe over 2000 different entities. Most interesting comparison queries on models of this size (identification of specific changes as illustrated above) run in under 5 seconds on a modern PC.

5. Related Work

Several other research efforts are concerned in one way or another with comparing model variants syntactically and providing an adequate representation for the resulting differences. For space reasons, we limit our discussion to only some of them.

[6] and [7] deal with comparing various types of UML models. The focus of these works is to supporting software development in the context of the Model Driven Architecture. To our knowledge, they do not attempt to describe changes generically, as our approach does.

[8] provides an ontology and a set of basic formal definitions related to the comparison of RDF graphs. [9] and [10] describe two systems currently in development that allow for efficiently storing a potentially large number of versions of an RDF model by using a compact representation of the raw changes between them (similar to our comparison model.) These works concentrate on space-efficient storage and transmission of change sets, but do not go into depth regarding how to use them to support identification of changes at a higher level of abstraction.

Finally, an extensive base of theoretical work is available from generic graph comparison research (see [2]). As argued in Section 2, we consider this work generally complementary to ours, and appropriate for a different set of problems.

6. Summary and Future Work

Despite its high practical relevance, the model comparison problem has only recently started to receive the attention it deserves. We are proposing Delta-P, an approach to model comparison that relies on Semantic Web notations and techniques. This approach comprises three main steps, namely converting models to the RDF notation, producing a raw comparison model that contains basic comparison results, and analyzing this model through queries to identify changes at a higher level of abstraction. Due to this structure, the approach is capable of supporting multiple schemata, allowing for identifying changes specific to each one of them.

Delta-P can only work properly on models that assign unique, stable identifiers to relevant model entities. Although, in theory, this characteristic may appear too restrictive, most actual modeling tools support the feature, a fact that makes our approach useful in a wide variety of practical applications. We have also implemented Delta-P in Evolyzer, a tool for analyzing the evolution of software process models. Our work with the V-Modell has shown that our approach is viable and efficient even on complex models.

Having a solid comparison system is only the first step towards a complete model versioning environment. Particularly, good support for collaborative work on models requires also a complementary merge algorithm. We have done initial work on extending our algorithms to support comparisons with common ancestor. Our experience so far, shows that it is possible to also use queries to specify a variety of relevant merge conflicts. This way, a merge is done by simply uniting the models and running a number of queries that identify possible conflicts.

Additional ongoing work is concentrated in turning our database based system into a full flexed model versioning system. An initial database containing about 600 development versions of the V-Modell, shows us that the approach is viable from a performance point of view. We are looking forward to building an operational prototype versioning system in the following months.

Acknowledgments. This work was supported in part by the German Federal Ministry of Education and Research (V-Bench Project, No. 01| SE 11 A).

7. References

[1] Soto, M., Münch, J.: Process Model Difference Analysis for Supporting Process Evolution. In: Proceedings of the 13th European Conference in Software Process Improvement, EuroSPI 2006. Springer LNCS 4257 (2006)

[2] Kobler, J., Schöning, U., Toran, J.: The Graph Isomorphism Problem: Its Structural Complexity. Birkhäuser (1993)

[3] Manola, F., Miller, E. (eds.): RDF Primer. W3C Recommendation, available from http://www.w3.org/TR/rdf-primer/ (2004) (last checked 2006-03-22)

[4] Prud'hommeaux, E., Seaborne, A. (eds.): SPARQL Query Language for RDF. W3C Working Draft, available from http://www.w3.org/TR/rdf-sparql-query/ (2006) (last checked 2006-10-22)

[5] V-Modell XT. Available from http://www.v-modell.iabg.de/ (last checked 2007-02-16).

[6] Alanen, M., Porres, I.: Difference and Union of Models. In: Proceedings of the UML Conference, LNCS 2863 Produktlinien. Springer-Verlag (2003) 2-17

[7] Lin, Y., Zhang, J., Gray, J.: Model Comparison: A Key Challenge for Transformation Testing and Version Control in Model Driven Software Development. In: OOPSLA Workshop on Best Practices for Model-Driven Software Development, Vancouver (2004)

[8] Berners-Lee, T., Connolly D.: Delta: An Ontology for the Distribution of Differences Between RDF Graphs. MIT Computer Science and Artificial Intelligence Laboratory (CSAIL). Online publication http://www.w3.org/DesignIssues/Diff (last checked 2006-03-30)

[9] Völkel, M., Enguix, C. F., Ryszard-Kruk, S., Zhdanova, A. V., Stevens, R., Sure, Y.: SemVersion - Versioning RDF and Ontologies. Technical Report, University of Karlsruhe. (2005)

[10] Kiryakov, A., Ognyanov, D.: Tracking Changes in RDF(S) Repositories. In: Proceedings of the Workshop on Knowledge Transformation for the Semantic Web, KTSW 2002. Lyon, France. (2002)