

On-Demand Merging of Traceability Links with Models

Dimitrios S. Kolovos, Richard F. Paige and Fiona A.C. Polack

Department of Computer Science, The University of York, UK, York, YO10 5DD
{dkolovos, paige, fiona}@cs.york.ac.uk

Abstract. Model management operations such as refinement and transformation modify existing or produce new models. To assist both engineers and tools in tracing the path of changes and decisions, maintenance of traceability information is essential. There are two main approaches for storing traceability; in the first, traceability information is embedded inside the models, while in the second, traces form separate models. The former allows users to inspect traceability information in a visual manner, but pollutes the models, while the latter keeps the models clean, but visualization is troublesome. In this paper, we propose a unification of the two approaches via model merging. We suggest that traceability information should be maintained in separate models, which can be merged with the primary model(s) on demand to produce annotated models for inspection purposes. We present a concrete example that demonstrates the practicality of our approach.

1 Introduction

Models are becoming increasingly important to software development. It is common practice that during a project life-cycle, a significant number of models are created and modified, mostly manually. Moreover, with the advent of Model Engineering infrastructure, models are also managed using automated facilities such as transformation engines. To allow both engineers and tools to trace the path of decisions and changes, maintaining traceability information is essential. The term *traceability information* applies to a wide spectrum, ranging from hand-crafted intra-model links, such as stereotyped UML associations, to tool-generated inter-model links relating elements of the source and target models of an automatic transformation.

The rest of the paper is organized as follows: In Section 2, we discuss two alternatives for storing and managing traceability information and identify points in favour and against for each. In Section 3, we propose a technique based on *model merging* for combining the strong points of the two approaches while eliminating their weaknesses. In Section 4, we present a concrete working example that demonstrates the feasibility and practicality of our proposal and in Section 5 we conclude and identify interesting further work on the subject.

2 Background

There are two approaches for storing and managing traceability links. In the first, links are embedded inside the models they refer to in the form of new model elements, while in the second, traceability links are stored in separate models.

2.1 Embedded Traceability Links

Under this approach, traceability links are embedded in the models they refer to in the form of new model elements. For example, a common practice to represent *refinement* traceability links in UML models is to add stereotyped (e.g. <<refines>>) associations that connect the participating elements [1]. This approach is popular with modellers for its human-friendliness as it represents traceability links as visual model elements that people can easily inspect and navigate.

On the other hand, this approach inherently applies only to representing intra-model traceability links. Moreover, embedding growing amounts of traceability links inside a model causes *pollution* [2] of the model. We have to bear in mind that traceability is only one of the many concerns when modelling, and embedding information relevant to all concerns inside the model will gradually render it overcrowded with elements of secondary importance. Another issue is *uniformity*; if traceability links are represented in ad-hoc ways (e.g. using different types of associations and ad-hoc names for stereotypes in UML), it becomes difficult to distinguish them from the other model elements. This renders automatic processing by tools particularly challenging. To contribute to *uniformity*, in [3], a UML profile for encoding traceability in UML models using a set of well-defined stereotypes is discussed.

2.2 External Traceability Links

In this approach, traceability links are stored externally to the models they refer to (i.e. in a separate model). A requirement of this approach, is that each element of the related model(s) must have a unique and persistent identification feature (such as the *xmi.id* identifier in MOF and EMF), so that external links to it can be resolved unambiguously.

If links are to be stored in an external model, an essential step is to define the metamodel to which this traceability model will conform. The simplest approach is to define a custom metamodel for each type of traceability that needs to be stored. For example, in [4], a simple metamodel tailored to transformation traceability is presented. A more sophisticated approach is to define a generic metamodel, such as the *Unified Traceability Scheme* discussed in [5]. This metamodel, with proper extensibility mechanisms, should be able to encode any type of traceability links. Finally, another option is to define a core traceability metamodel that encodes a basic set of common features and then define extensions for different types of traceability links. Although this can be

implemented with generic weaving tools such as the ModelWeaver [6], the semantics of *metamodel extensions* are not yet clearly defined. Further discussion on the advantages and disadvantages of a generic metamodel versus a small core that can be extended for each traceability domain can be reduced to a general discussion about generic modeling languages (such as the UML), versus domain specific languages, a subject clearly outside the scope of this paper.

Regardless of the traceability metamodel, an advantage of storing traceability links in separate models is that it facilitates loose coupling between the models and the links, thus keeping the models *clean*. Moreover, id-based links encoded in a metamodel with well-defined semantics are easier to manage and analyse with automated tools.

By contrast, id-based links are not particularly helpful to humans who need to inspect the models. Taking into consideration that a major role of traceability links is to assist modellers in decision making, this is considered to be a major downside of this approach.

3 On-Demand Merging

Although both embedded and external traceability approaches have disadvantages, the external traceability model is more flexible, as it is usable for managing both intra-model and inter-model traceability information. In this section, we propose an approach to external traceability that overcomes its primary problem, that of lack of human-friendliness.

To achieve this, we propose automated merging of traceability links stored in separate models with the models to which they refer. This can produce models annotated with traceability elements *on-demand*. We use the term *on-demand* to stress the fact that in each occasion only a subset of the traceability information needs to be visualized and therefore the merging process should be selective and customizable. Moreover, since there is no consensus on a global traceability metamodel yet, the solution we propose must apply to all possible traceability metamodels. Also, to achieve generality, it must not be limited to the context of a single modelling language (e.g. UML).

For merging models with traceability links, we use the Epsilon Merging Language (EML) [7]. EML is a rule-based language for merging models of diverse metamodels and technologies. In EML, model merging is performed in two phases. In the *matching* phase, correspondences between elements of the source models are established. In the *merging* phase, matching elements identified in the previous phase are merged, while elements with no matching opposites are optionally transformed into elements in the target model.

From a tool-support perspective, as discussed in [7], the execution engine of EML supports managing EMF[8] and MOF (MDR)[9] models as well as XML documents. This renders our approach practical for models created with any modelling tool that has XMI or XML exporting capabilities. Moreover, development tools for constructing, debugging and executing EML specifications have

been implemented as plug-ins for the Eclipse[10] open-source framework, and are publicly available at [11].

Merging traceability links with models is just one scenario of use of EML. In the matching phase of this scenario, a desired subset of links is matched with the elements they refer to via the persistent identities. In the merging phase, pairs of matching links and elements are merged to form annotated elements in the target model. Since we wish the target model to be an annotated copy of the source model, we attach the *LeftAndTargetCommonMetamodelMergingStrategy* merging strategy and the *CommonMetamodelTransformationStrategy* transformation strategy [7] to the EML specification to relieve the user from the task of composing EML rules that perform deep-copies of elements from the original to the annotated model.

The *LeftAndTargetCommonMetamodelMergingStrategy* strategy merges each element from the left model with matching elements from the right model into a deep-copy of the left element in the target model.

The *CommonMetamodelTransformationStrategy* strategy creates deep copies of the non-matching elements of the source model in the target model. User-defined EML merge and transform-rules can override the attached merging or transformation strategy respectively or extend it via the use of the *auto* keyword. In the case of extension, the behaviour of the strategy is executed before the body of the rule.

The functionality of the *strategies* presented here is further discussed in the case study that follows. However, for a complete discussion on advanced aspects of EML such as *strategies* and rule-extension, readers can refer to [7].

4 Case Study

In this section, we provide a concrete example that demonstrates the practicality of our approach. In our scenario, we have used EML to merge two UML 1.4 models such as the *Hotel* and *CarRental* models displayed in Figure 2 and 3 into a model named *Merged*, displayed in Figure 4. A side effect of the EML merging process is the production of a traceability model (*Trace*) that stores traceability links and conforms to the MOF-based metamodel displayed in Figure 1.

By encoding traceability links in a separate model, *Merged* remains clean and tidy. However, by inspection it is not always clear which classes originated from the first model (*Hotel*), which from the second (*CarRental*) and which existed in both. A desirable visualization scheme would be for each class in the merged model to contain a stereotype named after the model from which it originated or named “common” if it existed in both models.

To visualize this information, we need to merge the *Trace* model with the *Merged* model, into an *Annotated* model. Merging is performed via the EML specification displayed in 1.1.

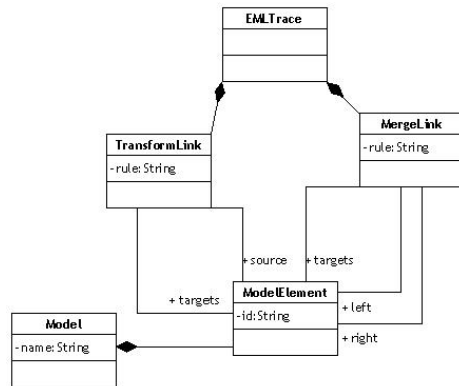


Fig. 1. EML Trace Metamodel

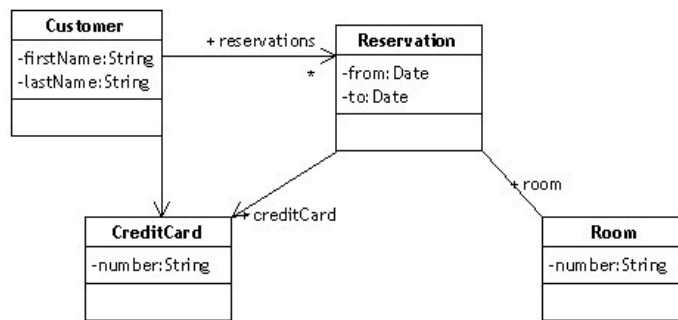


Fig. 2. Hotel model

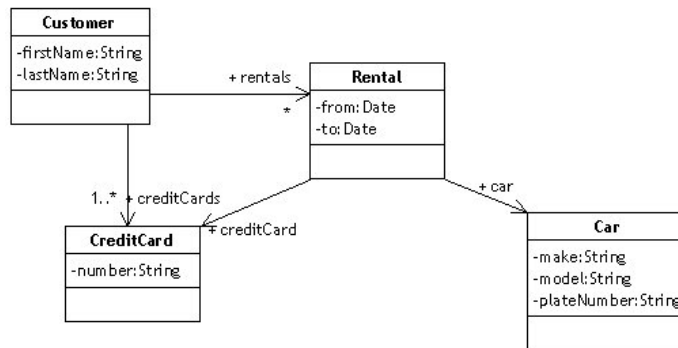


Fig. 3. Car Rental model

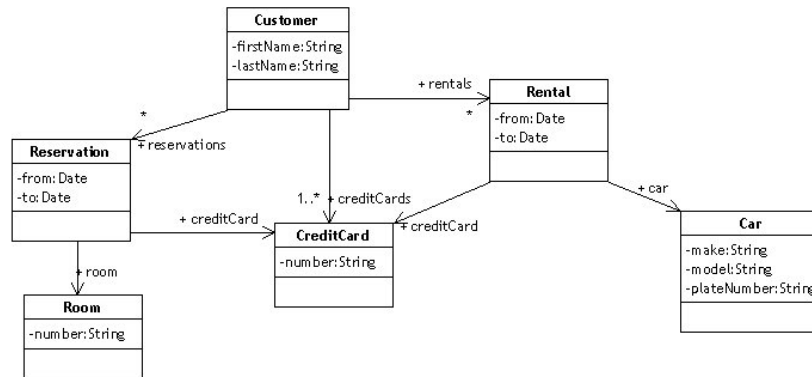


Fig. 4. Result of merging the Hotel and Car Rental models

Listing 1.1. EML Specification merging the Merged model with the Trace Model

```

pre {
  def commonStereotype : new Annotated!Stereotype;
  commonStereotype.name := 'Common';
}

rule ClassWithTransformationLink
  match c : Merged!Class
  with l : Trace!TransformationLink {

    compare {
      return l.targets.exists(t|t.id = c.id());
    }

  }

rule ClassWithMergeLink
  match c : Merged!Class
  with l : Trace!MergeLink {

    compare {
      return l.targets.exists(t|t.id = c.id());
    }

  }

auto rule Class2TransformedClass
  merge c : Merged!Class
  with l : Trace!TransformationLink
  into ac : Annotated!Class {

    def st : Annotated!Stereotype;
    def existing : Sequence(Annotated!Stereotype);
    def modelName : String;

    modelName := l.source.owner.name;
    existing := Annotated!Stereotype.allInstances()
      .select(s|s.name = modelName);

    if (existing.size() > 0){
      st := existing.first();
    }
    else {
  
```

```

    st := Annotated!Stereotype.newInstance();
    st.name := modelName;
    st.namespace := c.namespace.equivalent();
  }
}

auto rule Class2MergedClass
merge c : Merged!Class
with l : Trace!MergeLink
into ac : Annotated!Class {

  ac.stereotype.add(commonStereotype);

}

post {
  commonStereotype.namespace :=
    Merged!Model.allInstances().first().equivalent();
}

```

In the *pre* section of the EML specification, a stereotype (*commonStereotype*) is created in the *Annotated* model. This stereotype will be later on attached to classes that existed in both models. Note that the namespace of the stereotype cannot be assigned yet since the *Annotated* model is yet empty.

Match-rule *ClassWithMergeLink* matches a class of the *Merged* model against a merge-link from the *Trace* model. To match, the persistent identity of the class (retrieved via the built-in *id()* operation) must match with one of the identities of the targets of the link. Identical is the matching logic for the *ClassWithTransformLink* match-rule that matches classes against transform-links.

Classes that have been matched with transform-links indicate that they were part of only one of the models originally merged. In merge-rule *Class2TransformedClass*, each such class is merged with the matching transform-link. Since the rule is indicated as *auto*, execution of its body is preceded by execution of the behaviour specified by the attached merging strategy (*LeftAndTargetCommonMetamodelStrategy*). Therefore, before the body of the rule is executed, the source class *c* has been deeply-copied into the target class *ac* in the *Annotated* model.

The body of the rule states that once the deep-copy has been performed, an additional stereotype must be attached to *ac*. The name of the stereotype will be the name of the model the class originated from (retrieved via the trace link by *l.owner.name*). If the stereotype has not already been created in another execution of the rule, it is created and its namespace is set to the *equivalent* of the namespace of the class in the *Merged* model. For, details about the functionality of the *equivalent()* built-in operation see [7].

Similarly, classes that have been matched with merge-links indicate that they existed in both of the models originally merged. In merge-rule *Class2MergedClass*, each such class is merged with its matching merge-link. As the rule is declared as *auto*, the body of the merging strategy is executed and after that, in the rule's body it is declared that the *commonStereotype* defined in the *pre* section must be attached to the stereotypes of the *ac* class in the *Annotated* model.

Finally, in the *post* section that is executed after merging has been completed, we set the namespace of the *commonStereotype* defined in the *pre* section. The

Annotated model that is produced by merging the *Merged* and *Trace* models by executing this EML specification is displayed in Figure 5.

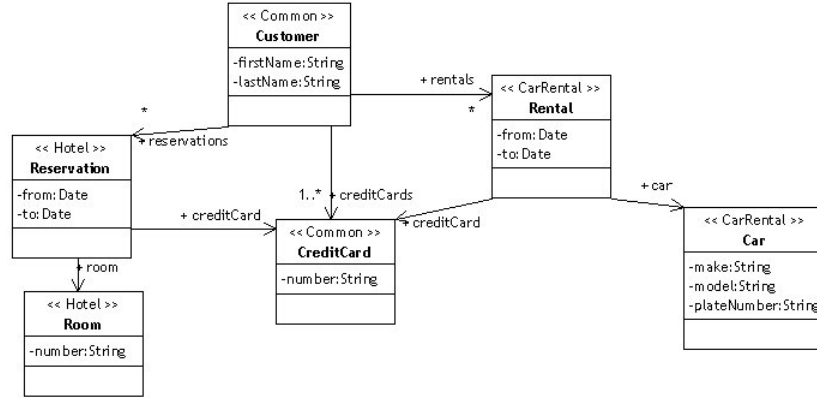


Fig. 5. Result of merging the model of Figure 4 with trace links into an annotated model

This example has illustrated that our approach is independent of the modelling language since EML treats UML as any other MOF-based language. It is also independent of the traceability scheme since instead of the custom traceability metamodel displayed in Figure 1, any other metamodel could have been used. Finally, it is particularly flexible as the specification of the visualization of traceability in the *Annotated* model is entirely encoded in the EML specification of Listing 1.1.

5 Conclusions and Further Work

In this paper, we have proposed a novel approach to on-demand merging of traceability links with models, to produce models annotated with traceability information. As demonstrated through the case study, using EML as a merging language renders our approach particularly flexible since it applies on models and traceability schemes of diverse metamodels and technologies.

An interesting subject for future work is the reverse process: that of *cleaning* already *polluted* models with overwhelming traceability information, possibly by using a transformation language to transform a *polluted* model into a pair consisting of a *clean* model and one (or more) traceability models.

6 Acknowledgements

The work in this paper is partially supported by the European Commission via the MODELWARE project, co-funded under the “Information Society Technologies” Sixth Framework Programme (2002-2006). Information included in this document reflects only the authors views. The European Commission is not liable for any use that may be made of the information contained herein.

References

1. William Heaven and Anthony Finkelstein. A UML profile to support requirements engineering with KAOS. *IEE Proceedings: Software*, 151(1):10–27, 2004.
2. Future Research Topics Discussion. Traceability Workshop, EC-MDA, November 2005. <http://www.sintef.no/upload/10558/Future-Research-Topics.pdf>.
3. Bert Vanhooff and Yolande Berbers. Supporting Modular Transformation Units with Precise Transformation Traceability Metadata. In *Proc. Traceability Workshop, European Conference in Model Driven Architecture (EC-MDA)*, 2005.
4. Frederic Jouault. Loosely Coupled Traceability for ATL. In *Proc. Traceability Workshop, European Conference in Model Driven Architecture (EC-MDA)*, pages 29–37, 2005.
5. Angelina E. Limon and Juan Garbajosa. The Need for a Unifying Traceability Scheme. In *Proc. Traceability Workshop, European Conference in Model Driven Architecture (EC-MDA)*, pages 47–55, 2005.
6. Marcos Didonet Del Fabro, Jean Bezivin, Frederic Jouault, Erwan Breton, Guillaume Gueltas. AMW: A Generic Model Weaver. In *Proceedings of IDM05*, 2005.
7. Dimitrios S. Kolovos, Richard F. Paige and Fiona A.C. Polack. Merging Models With the Epsilon Merging Language (EML). In *Proc. ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (Models/UML 2006)*, Genova, Italy, October 2006. LNCS. (to appear).
8. Eclipse.org. Eclipse Modelling Framework. <http://www.eclipse.org/emf>.
9. Sun Microsystems. Meta Data Repository. <http://mdr.netbeans.org>.
10. Eclipse Foundation, Official Web-Site. <http://www.eclipse.org>.
11. Dimitrios S. Kolovos. Extensible Platform for Specification of Integrated Languages for mOdel maNagement (Epsilon), Official Web-Site. <http://www.cs.york.ac.uk/~dkolovos/epsilon>.