Higher-Order Transformations in AToMPM

Simon Van Mierlo

University of Antwerp

1 Introduction

Higher-Order Transformations, or HOTs, are transformations which receive a model of a transformation as input and produce a model of a transformation as output [2]. A specific use-case for HOTs is the evolution of domain-specific languages, where HOTs are constructed to co-evolve existing transformations together with their domain- or target language. This document describes how higher-order transformations are supported in AToMPM. In AToMPM, RAMification [1] is performed on the original metamodel of a language, which results in the pattern languages used to contruct to patterns which appear in the rules of a transformation. As a first step, this process will be modelled explicitely as a transformation on the abstract and concrete syntax models of a language, which is outlined in Section 2. In 3, the changes made to the transformation engine are explained. To conclude, current issues with the RAMification transformation and transformation back-end are listed as a reference for future work.

2 Explicitely Modeling RAMification

RAMification is the process of modifying the metamodel of a language in such a way that it can be used in the patterns of transformation rules. It consists of three adaptation phases:

1. **Relaxation**, which relaxes constraints on the language which are no longer valid for the pattern language. For instance, it should be possible to instantiate abstract classes in rule patterns.
2. **Augmentation**, which augments metamodel elements with attributes used by the transformation engine.

3. **Modification**, which modifies the data type of attributes such that they can express constraints on attribute values or actions which compute the new value of the attribute.

In AToMPM, this is implemented by a hard-coded routine which operates directly on the compiled metamodel. However, the RAMification process can be encoded as a transformation which takes as input the models describing the abstract and concrete syntax of the original language and transforms these in such a way that it results in the models describing the abstract and concrete syntax of the pattern language. An advantage to this approach is that the RAMification process is made explicit, which makes it easier to discover faults and adapt the transformation. In Figure 2 the transformation, as it appears in the AToMPM model editing environment, is shown. In detail, these are the steps taken by the transformation, followed by the letter which denotes to which phase in the RAMification process they belong (R, A or M):

- **disableAction** and **disableConstraint** (R): Disables global actions and constraints defined on the metamodel.
- **removeLowerBound** (R): Sets the lower bound of multiplicites to 0.
- **renameClass**, **renameAssociation** and **renameSelfAssociation** (M): Appends '__p' to classes and associations, which is needed for the transformation engine.
- **changeAttrType** (M): Changes the type of attributes to 'string', which allows conditions and actions to be specified on attribute values in patterns. Also appends '__' to attributes starting with '__'. This is because the __pLabel and __pMatchSubtypes attributes (introduced in one of the next steps) need to be scoped appropriately for HOTs. The first time a metamodel is ramified, each class will have a __pLabel and __pMatchSubtypes attribute. The next time, these attributes are renamed to ____pLabel and ____pMatchSubtypes, to allow a condition/action to be specified for the __pLabel and __pMatchSubtypes attributes which were introduced in the first RAMified metamodel.
- **makeConcrete** (R): Makes abstract classes concrete.
- **disableClassActionsConstraints** (R): Disables actions and constraints defined on the classes of the metamodel.

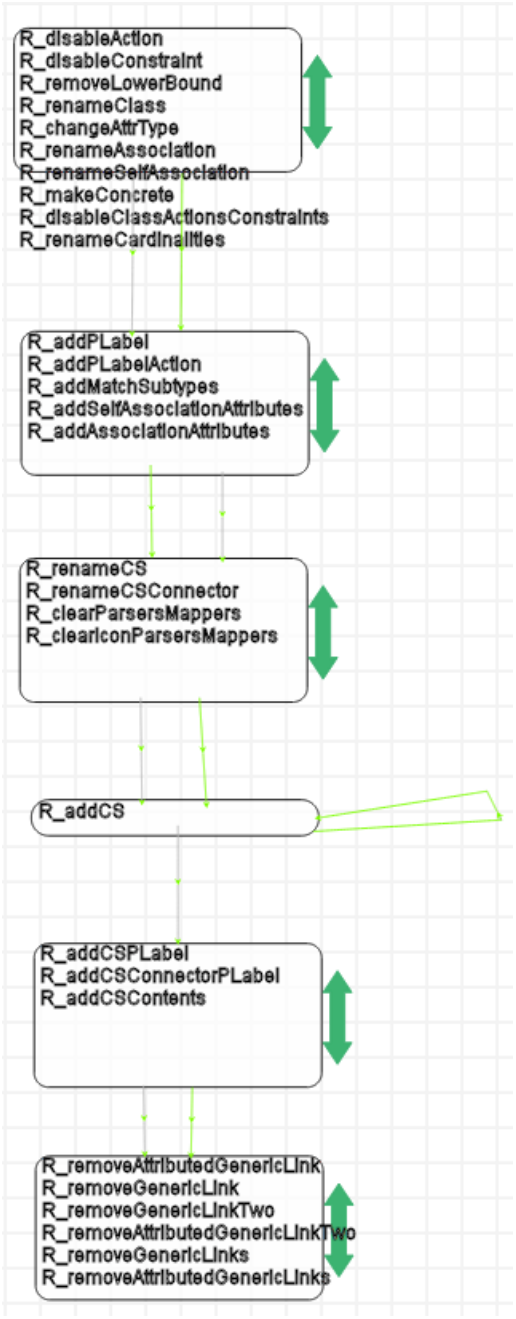

Fig. 1. RAMification Transformation in AToMPM

- **renameCardinalities** (M): Modifies the cardinalities such that they reflect the change in name of the association.
- **addPLabel** (A): Adds the __pLabel attribute to classes, which is used to identify elements across patterns of a rule.
- **addPLabelAction** (A): Adds the action which automatically assigns the __pLabel attribute when a class is instantiated.
- **addMatchSubtypes** (A): Adds the __pMatchSubtypes attribute, which signifies whether instances of the subclasses of an element of a pattern should also be matched.
- **addSelfAssociationAttributes** and **addAssociationAttributes** (A): Adds the __pMatchSubtypes and __pLabel attributes to associations.
- **renameCS** (M): Renames the concrete syntax icon of a class to match its new name.
- **renameCSConnector** (M): Renames the concrete syntax connector of an association to match its new name.
- **clearParserMappers** and **clearIconParserMappers** (M): Disables parsers and mappers of classes and icons.
- **addCS** (A): Adds a concrete syntax icon for (former) abstract classes.
- **addCSPLabel** and **addCSConnectorPLabel** (A): Adds a concrete syntax representation for the __pLabel attribute.
- **addCSContents** (A): Adds a concrete syntax representation to the icon introduced for (former) abstract classes.
- **remove*Link***: Removes GenericLinks introduced during the transformation.

To create a pattern language, a modeller loads both the models of the abstract and concrete syntax of the modelling language. He then executes the RAMification transformation and compiles the resulting model first to an abstract syntax metamodel called <**OriginalName**>**.ramified.metamodel** and a concrete syntax metamodel called <**OriginalName**>**.ramified.defaultIcons.metamodel**. The concrete syntax metamodel can then be loaded as a toolbar in AToMPM to be used in patterns.
As explained, this transformation is capable of both RAMifying the metamodel of a modelling language, as well as its RAMified version(s). It is as such possible to model higher-order transformations.

However, to make sure rules which use these pattern models are executed correctly by the transformation engine, some modifications had to be made. These are outlined in the next section.

3 Modifying the Transformation Engine

The following changes were made to AToMPM to accomodate for the changes in the RAMification process:

1. The contraint on which elements can appear in patterns has been relaxed: elements which belong to a *.ramified.metamodel metamodel are accepted besides *.pattern.metamodel. This change had to be made in the contraints on the Transformation metamodel, as well as in the code of the transformation back-end, which only considered *.pattern.metamodels.
2. Handle multiple RAMifications of __pLabel and __pMatchSubtypes attributes. This basically introduces scoping, which had to be added in the code of the transformation back-end.

4 Issues and Future Work

These are the current issues with the RAMification transformation and the use of the RAMified metamodels:

1. The concrete syntax that is added for abstract classes is 'stretched' because the icon and its content are placed on a different position. This results in a bounding box which is too large.
2. When RAMifying an already RAMified metamodel, the concrete syntax of the __pLabel attribute should appear on another position than the previously introduced concrete syntax for the attribute which is now called ____pLabel.
3. Metamodels aren't loaded automatically when a transformation has to create an instance of a class of a metamodel which isn't loaded already.

References

1. Thomas Kühne, Gergely Mezei, Eugene Syriani, Hans Vangheluwe, and Manuel Wimmer. Explicit transformation modeling. In Sudipto Ghosh, editor, *Models in Software Engineering*, volume 6002 of *Lecture Notes in Computer Science*, pages 240–255. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-12261-3_23.

2. Massimo Tisi, Frdric Jouault, Piero Fraternali, Stefano Ceri, and Jean Bzivin. On the use of higher-order model transformations. In Richard Paige, Alan Hartman, and Arend Rensink, editors, *Model Driven Architecture - Foundations and Applications*, volume 5562 of *Lecture Notes in Computer Science*, pages 18–33. Springer Berlin / Heidelberg, 2009.