Plug & Play Model Transformations - A DSL for Resolving Structural Metamodel Heterogeneities[*]

M. Wimmer
TU Vienna
wimmer@big.tuwien.ac.at

G. Kappel
TU Vienna
gerti@big.tuwien.ac.at

A. Kusel
JKU Linz
kusel@bioinf.jku.at

W. Retschitzegger
JKU Linz
werner@bioinf.jku.at

J. Schoenboeck
TU Vienna
schoenboeck@bioinf.jku.at

W. Schwinger
JKU Linz
wieland@jku.at

ABSTRACT

Model transformations play a key role in the vision of Model-Driven Engineering. Thereby, the resolution of structural heterogeneities between metamodels (MMs) represents *the* key challenge. For this task, current approaches require the definition of partly tricky, low-level recurring transformation logic but neglect to offer reusable components. Moreover, little attention has been paid to heterogeneities caused by the concept of inheritance, although extensively used in MMs. Therefore, we propose to specify model transformations in a plug & and play manner by a set of predefined mapping operators (MOps) representing a DSL to resolve structural heterogeneities. For coping with inheritance in MMs, we introduce an inheritance mechanism between MOps allowing to reuse parts of the mapping definitions. Moreover, dedicated MOps for resolving heterogeneities when one MM comprises inheritance hierarchies whereas the other one does not are presented, which are well-known problems in object-relational transformations and object-oriented refactorings.

Categories and Subject Descriptors

D.2.13 [**Software Engineering**]: Reusable Software

General Terms

Languages

Keywords

Model Transformation, Reusability, Inheritance

1. INTRODUCTION

Model-Driven Engineering (MDE) places models as first-class artifacts throughout the software lifecycle [4] whereby model transformations play a vital role. In the context of transformations between different metamodels (MMs) and their corresponding models, the overcoming of *structural heterogeneities*, being a result of applying different metamodeling constructs for the same semantic modeling concept is *the* key challenge. As a simple case study Fig. 1 shows two MM extracts of the UML ActivityDiagram versions 1.4 and 2.2 from this year's Transformation Tool Contest[1] serving as a running example throughout the paper. Whereas in version 1.4 (cf. left-hand side (LHS) of Fig. 1) different kinds of control nodes are represented by the attribute `Pseudostate.kind`, the semantically equivalent information is represented in version 2.2 (cf. right-hand side (RHS) of Fig. 1) by a type hierarchy, i.e., explicit classes inheriting from `ActivityNode` are available.

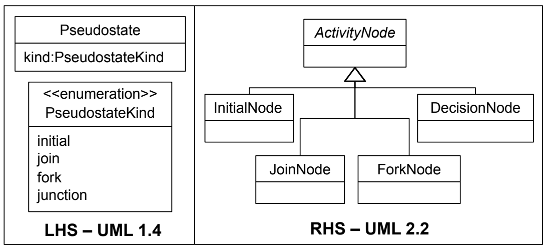


Figure 1: UML Activity Diagram Versions

As the example indicates, a semantic concept in one MM can be expressed by any of the common core concepts of semantic data models [8], i.e., classes, attributes, references, and inheritance, in the other MM resulting in manifold heterogeneities [3, 13]. For their resolution, current best practices comprise specifying model transformations between different MMs having to deal with all the *low-level intricacies* of a certain transformation language. Moreover, current transformation languages *lack appropriate reuse mechanisms.*

We therefore propose to specify model transformations by means of *abstract mappings* using a set of reusable transformation components, called *mapping operators* (MOps) to resolve recurring structural heterogeneities. The rationale behind is to follow an MDE-based approach, since abstract mappings can be seen as platform-independent transformation models specified using a predefined library of MOps, resembling, in terms of MDE, a DSL to resolve structural heterogeneities. Although abstracting from the underlying transformation language, abstract mappings can be automatically translated to different platforms, i.e., transformation languages by means of higher-order transformations (HOTs) [18] since the MOps exhibit a clearly defined operational semantics. An initial set of MOps has been presented in [19] focusing on the resolution of heterogeneities between classes, attributes and references, only. Based on this on work we present in this paper first an inheritance mechanism between MOps allowing to reuse parts of the mapping definitions for coping with MM inheritance. Secondly, dedicated inheritance MOps (iMOps) for resolving

[*]This work has been funded by the Austrian Science Fund (FWF) under grant P21374-N13.

[1]http://www.planet-research20.org/ttc2010/

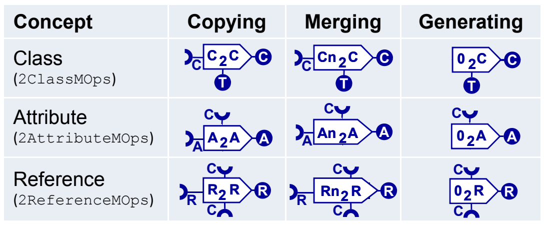
heterogeneities when one MM uses inheritance whereas the other expresses the same semantic concepts without the usage of inheritance. The motivation behind this work is that inheritance is extensively used in MMs as, e.g., the evolution of the UML standard reveals [14]. Thereby, substantial changes concerning the inheritance hierarchies as well as the application of abstract classes have taken place, emphasizing the need for support to resolve inheritance heterogeneities.

The remainder of this paper is structured as follows. Whereas Section 2 revisits the MOps already proposed in [19], Section 3 introduces an inheritance mechanism between MOps and Section 4 presents dedicated iMOps for resolving heterogeneities caused by inheritance. The subsequent Section 5 gives a brief overview on the prototypical implementation of the approach. Finally, Section 6 surveys related work and Section 7 concludes this paper with a critical discussion as well as an outlook on future work.

2. MOps IN A NUTSHELL

We briefly revisit our MOps [19] resolving heterogeneities wrt. to classes, attributes and references. The presented MOps can be divided into so-called *kernel MOps* and *composite MOps* building the basis for the newly introduced iMOps in the subsequent section.

Kernel MOps. In order to provide a MOps kernel, i.e., a minimal set of required MOps to overcome structural heterogeneities in-the-small, we systematically combined the core concepts of semantic data models, being (i) classes, (ii) attributes, and (iii) references with different mapping cardinalities, thereby complementing the work of Legler and Naumann [13] focusing on attributes only. Therefore, MOps are provided for the basic tasks of a model transformation, being (i) *copying*, (ii) *merging* and (iii) *generating*[2]. These tasks are supported for all the core concepts of semantic data models as can be seen in Fig. 2. In this respect, a `C2C` operator is used to map a LHS class to a RHS class and expresses that each instance of the LHS class is copied to an according instance of the RHS class. In contrast, a $\mathtt{C}^n_2\mathtt{C}$ operator maps several LHS classes to a single RHS class, thus merging a certain combination of instances of the LHS classes to a single instance of the RHS class. Finally, a $\mathtt{0_2C}$ operator can be used in the situation that no LHS class is mappable to a RHS class thus simply generating instances of the RHS class. The described operational semantics for the `2ClassMOps` can be transferred analogously to the `2AttributeMOps` as well as to the `2ReferenceMOps`.



Concept	Copying	Merging	Generating
Class (`2ClassMOps`)	C2C	Cn2C	02C
Attribute (`2AttributeMOps`)	A2A	An2A	02A
Reference (`2ReferenceMOps`)	R2R	Rn2R	02R

Figure 2: Kernel MOps

As can be seen in Fig. 2, each MOp has *input ports* with required interfaces (left side of the component) as well as *output ports* with provided interfaces (right side of the component), typed to classes (C), attributes (A), and references (R) in terms of Ecore[3] types. Thus, *metamodel independence* is achieved, i.e., MOps can be reused between arbitrary Ecore-based MMs. Since there are dependencies between MOps, e.g., a link can only be set after the two objects to be connected are available, each `2ClassMOp` additionally offers a *trace port* (T) at the bottom, providing context information, i.e., offering information about which output objects have been produced from which input objects. This port can be used by other MOps to access context information via *required context ports* (C) with corresponding interfaces on top of the MOP, or in case of `2ReferenceMOps` via two ports, whereby the top port depicts the required source context and the bottom port the required target context (cf. Fig. 2). Since MOps are expressed as components, the transformation designer can apply them in a plug & play manner by binding their interfaces.

Composite MOps. Besides kernel MOps a set of *composite MOps* has been introduced in [19] encapsulating a larger recurring pattern of heterogeneity and therefore making the approach more scalable. In this respect, typical heterogeneities, e.g., *vertical partitioning*, i.e., attributes of a LHS class are distributed over several different RHS classes, have been modeled (cf. EBNF of the `VerticalPartitioner` shown in Table 1) allowing the transformation designer to benefit from this pre-modeled knowledge. Therefore, the concrete approach for specifying a mapping is as follows. The transformation designer detects a certain recurring heterogeneity and applies the corresponding composite MOp in the so-called *black-box view*. In this view, only classes are connected to the composite MOp, thereby providing an abstract high-level view on a model transformation. On switching to the *white-box view*, the *fixed parts* of a pattern get generated. Now, the transformation designer can complement the mapping by adding the *variable parts*, i.e., attribute and reference mappings, as required.

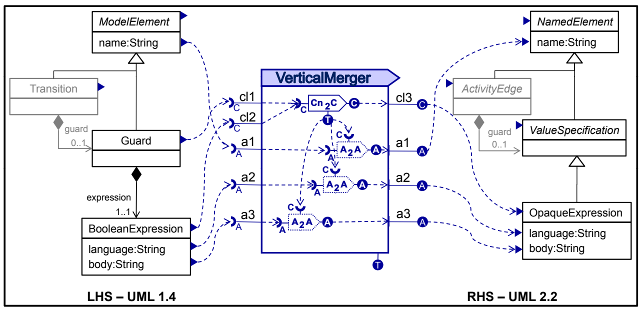


Figure 3: Whitebox-View of Composite MOp

For exemplifying this, Fig. 3 shows the application of the `VerticalMerger` composite MOp in the ActivityDiagram case study. Thereby the heterogeneity is resolved, that the guard of a transition is expressed by two separate classes `Guard` and `BooleanExpression` in UML 1.4, whereas it is expressed by a single class `OpaqueExpression` (inheriting from `ValueSpecification`) in UML 2.2. Thus, logic is needed, that is able to merge instances of the classes `Guard` and `BooleanExpression` to instances of the class `OpaqueExpression`. Therefore, the `VerticalMerger` composite MOp has been applied in the blackbox-view. When switching to the whitebox-view, the fixed parts of the composite MOp - being in this case the $\mathrm{C}^n_2\mathrm{C}$ according to the EBNF given in Table 1

[2]Please note that kernel MOps for splitting are not necessary being simply accomplished by several copying MOps.

[3]http://www.eclipse.org/modeling/emf/

Table 1: Composite MOps

MOp	Description	Composition of Kernel MOps (EBNF)
Copier	LHS class corresponds to exactly one RHS class, i.e., Copier MOp copies objects and its corresponding attributes and references	Copier = C_2C { A_2A | $A^n{}_2A$ | 0_2A | R_2R | $R^n{}_2R$ | 0_2R }.
VerticalPartitioner	LHS class corresponds to several RHS classes, i.e., VerticalPartitioner splits one source object into several target objects as well as its corresponding attributes and references	VerticalPartitioner = Copier { ObjectGenerator | Copier }.
VerticalMerger	Several LHS classes correspond to exactly one RHS class, i.e., VerticalMerger merges several source objects to one target object as well as its corresponding attributes and references	VerticalMerger = $C^n{}_2C$ { A_2A | $A^n{}_2A$ | 0_2A | R_2R | $R^n{}_2R$ | 0_2R }.
ObjectGenerator	No LHS class corresponds to an according RHS class, i.e., ObjectGenerator generates a new target object, as well as its corresponding attributes and references	ObjectGenerator = 0_2C { A_2A | $A^n{}_2A$ | 0_2A | R_2R | $R^n{}_2R$ | 0_2R }.

- get pre-generated. Now the transformation designer needs to complement the mapping by adding the variable parts as required. In this case three copying attribute MOps (cf. `A2A` in Fig. 3) are needed, that map the corresponding attributes. In order to further support the transformation designer in this step, a simple name matching of attributes and references takes place, which in this case leads to the automatic generation of the three required mappings since the names of the attributes match perfectly.

Although this solution allows to transform models correctly, it neglects the existence of the inheritance relationships between the classes and therefore has some main drawbacks. Firstly, the attribute mappings go beyond the scope of the mapped classes, as is the case with the attribute `name` which is defined in the classes `ModelElement` and `NamedElement` leading to confusing mapping specifications. Secondly, such attribute mappings which go beyond the scope of the class are likely to be redundant, since other classes also inheriting from some base class (like `ModelElement`) will probably re-specify such mappings. Thus, a mechanism is needed, that allows to reuse (inherit) such mappings. Finally, the same problem arises in the context of reference mappings. Therefore, in the following section an inheritance mechanism between MOps is introduced.

3. AN INHERITANCE MECHANISM BETWEEN MOps

After briefly revisiting existing MOps, we introduce an inheritance mechanism for MOps allowing to reuse parts of a mapping specification. The inheritance principle of object-oriented MMs is applied to MOps in the way that (i) a MOp between subclasses (*submapping*) inherits the feature mappings of a MOp between superclasses (*supermapping*) allowing for reuse and that (ii) a MOp between subclasses may refine the mapping specification by additional feature mappings. Thereby, a MOp might inherit from multiple other MOps, i.e., multiple inheritance is supported. Furthermore, mappings between superclasses can be set *abstract*, i.e., the mapping itself is not executable, needed in case the RHS class is abstract. On the execution level, the introduction of inheritance between MOps means that a mapping is executed additionally for all instances of subclasses which are not affected by a submapping. For being able to introduce an inheritance relationship between MOps certain preconditions must hold. Firstly, it has to be ensured that either the participating LHS classes of the supermappings and the submappings are in an inheritance relationship or all mappings have the same LHS class as input. The same constraint must hold on the RHS. These constraints must be ensured since the submappings inherit the feature mappings of the supermappings and therefore the features of the superclasses must also be available on instances transformed by the submappings.

To exemplify this, Fig. 4 depicts the example introduced in Fig. 3 resolved with inheritance between the mappings allowing to overcome the drawbacks raised in the previous section. Now, two abstract mappings have been added between the classes `ModelElement` and `NamedElement` and the classes `Guard` and `ValueSpecification`. This allows to keep the dependent mappings, i.e., attribute mappings and reference mappings in the scope of the mapping of the owned class, leading to well-structured mapping specifications. Moreover, dependent mappings can be inherited, removing redundancy in the mapping specification (e.g., `A2A` between `ModelElement.name` and `NamedElement.name` has to be specified only once in the example) and thus leading to easier maintainable and less error-prone mappings.

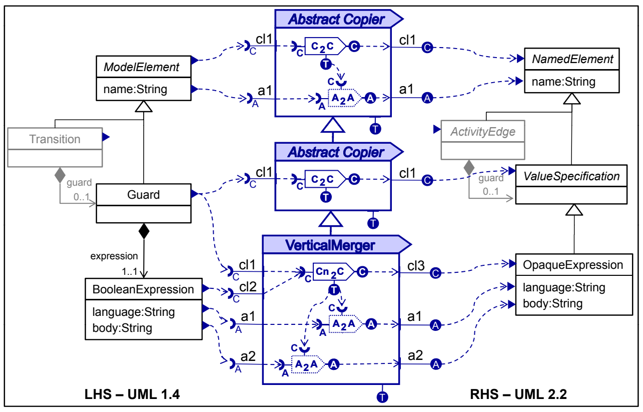


Figure 4: Whitebox-View with Inheritance

4. iMOps

After presenting an inheritance mechanism we now discuss iMOps resolving heterogeneities when one MM uses inheritance whereas the other one does not. The provided iMOps are summarized in Table 2 and represent well-known patterns from the area of object-relational transformations [11] and refactorings [7] (cf. *known uses* column). Due to space limitations we will only discuss the `A2I` (`Attribute2Inheritance`) in detail and briefly discuss the remaining iMOps.

A_2I iMOp. For resolving the heterogeneity that a LHS class offers type information by means of an attribute whereas the RHS MM provides explicit type information by means of an inheritance hierarchy an `A2I` iMOp is provided. An

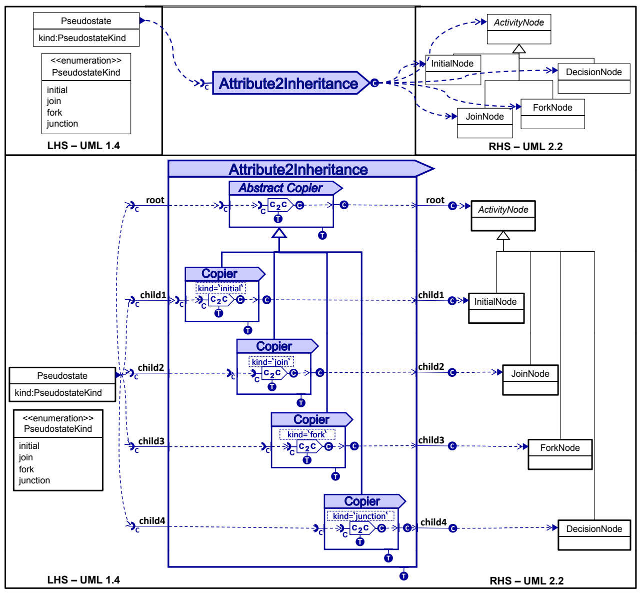


Figure 5: A_2I MOp

example of an *A2I* situation is depicted in Fig. 5 resolving the example described in Section 1 (cf. Fig. 1). Thereby, the transformation designer connects the single LHS class to the input port and the RHS classes of the inheritance hierarchy to the output ports in the blackbox-view (cf. top of Fig. 5). When switching to the whitebox-view (cf. bottom of Fig. 5), for each connected RHS class a `Copier` with a to be completed condition is pre-generated according to the EBNF shown in Table 2, whereby the `Copiers` have to inherit from each other. Thereby, the required conditions split the object set of the LHS according to the type information into subsets in the RHS corresponding to the subclasses of the inheritance hierarchy. When taking a look at the EBNF in Table 2, one can see that this pattern may not only consist of `Copiers` but of `VerticalPartitioners` and `ObjectGens` too. This is due to the fact, that an attribute in the LHS acting as input for MOps producing arbitrary RHS elements may carry type information, which might be encoded by an inheritance hierarchy in the RHS.

If we exchange LHS and RHS MMs of our example, i.e., the LHS MM provides explicit types whereas the RHS MM uses an attribute to represent the type information an $\mathtt{I_2A}$ iMOp can be applied, thus representing the opposite of the $\mathtt{A_2I}$ iMOp. In this case again `Copiers` could be used whereby the abstract `Copier` again maps the common attributes and the inheriting `Copiers` map the specific types. However, instead of the condition these `Copiers` require a MOp that generates the according attribute values, i.e., an $\mathtt{O_2A}$ MOp setting the according `Pseudostate.kind` value.

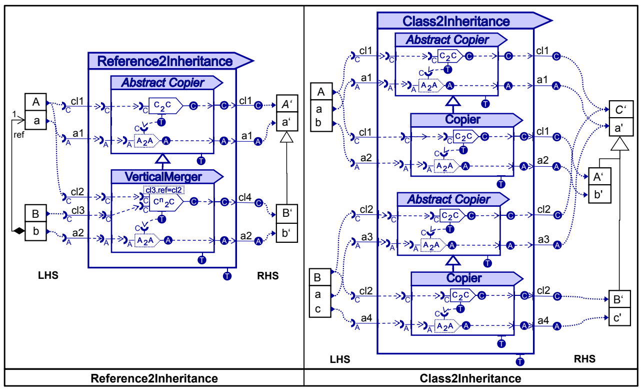


Figure 6: $\mathtt{R_2I}$ and $\mathtt{C_2I}$ MOp

R_2I and I_2R iMOps. For resolving the heterogeneity that a LHS MM uses delegation whereas the RHS MM uses inheritance the $\mathtt{R_2I}$ iMOp is provided as sketched in Fig. 6. Thereby the referenced LHS class `A` is mapped to the root of the RHS hierarchy `A'` by an (abstract) `Copier`. The referring LHS class `B` is mapped to the RHS class `B'` by a `VerticalMerger` which inherits from the base mapping, thereby reusing the attribute mapping between `a` and `a'`. For the case that the LHS MM uses inheritance whereas the RHS MM uses delegation the $\mathtt{I_2R}$ iMOp is provided.

C_2I and I_2C iMOps. A common scenario during MM evolution is that several independent classes are arranged in an inheritance hierarchy. To resolve this heterogeneity a $\mathtt{C_2I}$ iMOp is provided as sketched in Fig. 6. For each path of the RHS inheritance hierarchy a set of mappings inheriting from each other is required. Thus, the example exhibits two abstract `Copiers` with again one `Copier` inheriting from them. Nevertheless, no common mapping for the root class of the RHS inheritance hierarchy can be introduced since the attributes owned by the root class originate from different LHS classes. If an LHS inheritance structure is represented by independent RHS classes an $\mathtt{I_2C}$ iMOp can be applied whereby the MOps contained in the whitebox exhibit a condition which filters instances of a specific type only.

Table 2: Overview of iMOps resolving Non-Overlapping Heterogeneities

	iMOp	Description	iMOps EBNF	Known Uses
A	**A_2I** (Attribute2Inheritance)	LHS MM encodes type information by attribute values; RHS MM provides explicit types in an inheritance hierarchy	`A₂I = RootMapping {ChildMapping}.` `RootMapping = (Copier | VerticalMerger | ObjectGenerator).` `ChildMapping = (Copier | VerticalMerger | ObjectGenerator).`	Replace Type Code with Subclasses [7] One Inheritance Tree One Table [11]
	I_2A (Inheritance2Attribute)	LHS MM provides explicit types in an inheritance hierarchy; RHS MM encodes type information by attribute values	`I₂A = RootMapping ChildMapping {ChildMapping}.` `RootMapping = (Copier | VerticalMerger).` `ChildMapping = (Copier | VerticalMerger).`	Replace Subclass with Fields [7] One Inheritance Tree One Table [11]
R	**R_2I** (Reference2Inheritance)	LHS MM relates classes by references; RHS MM provides an explicit inheritance hierarchy	`R₂I = [RootMapping] ChildMapping {ChildMapping}.` `RootMapping = (Copier | VerticalMerger).` `ChildMapping = (VerticalMerger).`	Replace Delegation with Inheritance [7] One Class One Table [11]
	I_2R (Inheritance2Reference)	LHS MM provides explicit inheritance hierarchy; RHS MM relates classes by references	`I₂R = [RootMapping] ChildMapping {ChildMapping}.` `RootMapping = (Copier | VerticalMerger).` `ChildMapping = (VerticalMerger).`	Replace Inheritance with Delegation [7] One Class One Table [11]
C	**C_2I** (Class2Inheritance)	Independent LHS MM classes are contained in an inheritance hierarchy in RHS MM	`C₂I = [RootMapping] ChildMapping {ChildMapping}.` `RootMapping = (Copier | VerticalMerger).` `ChildMapping = (Copier | VerticalMerger).`	Extract Superclass [7] One Inheritance Path One Table [11]
	I_2C (Inheritance2Class)	LHS inheritance hierarchy is represented by independent RHS classes	`C₂I = [RootMapping] ChildMapping {ChildMapping}.` `RootMapping = (Copier | VerticalMerger).` `ChildMapping = (Copier | VerticalMerger).`	One Inheritance Path One Table [11]

5. PROTOTYPICAL IMPLEMENTATION

This section discusses our prototypical implementation, which bases on the Atlas Model Weaver (AMW) [6] as well as a higher-order transformation (HOT) for generating executable ATL code [10] out of the mappings.

Mapping Specification. The AMW framework, providing a generic infrastructure to declaratively specify (*plug*) mappings between two arbitrary MMs, has been accordingly extended with our MOps whereby for each MOp a direct or indirect subclass of `WLink` of the AMW MM is defined. This way the abstract syntax of our mapping DSL is defined, acting as input MM for the HOT. To ensure basic verification support of the specified mappings, constraints are provided using the EMF Validation Framework[4], e.g., every MOp has to be correctly connected to its source and target MM elements as well as to its context mappings.

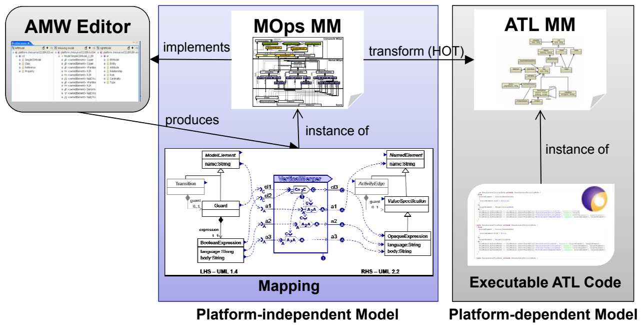


Figure 7: Prototypical Implementation

Mapping Execution. To execute (*play*) the *platform-independent* mappings, a mapping is automatically transformed to a concrete transformation language, representing the *platform-dependent* transformation model by means of a HOT (cf. Fig. 7). The HOT has to cope with kernel MOps only, since composite MOps and iMOps solely base on kernel MOps whereby each kernel MOp exhibits a clearly defined operational semantics [19]. Therefore, new user-defined composite MOps comprising kernel MOps can be added without the need of altering the HOT. In our prototype, ATL has been chosen as concrete transformation language. On examining the generated code our MOps abstract from ATL-specific intricacies as detailed in the following. For kernel MOps dealing with copying, partitioning and merging, declarative ATL code is generated in terms of matched rules. In case of MOps generating RHS elements, the provided declarative constructs miss expressivity, thus imperative code blocks need to be generated. Although ATL provides a basic rule inheritance mechanism, it is not able to (i) cope with multiple inheritance between rules and (ii) inherit between rules exhibiting different signatures, e.g., a matched rule merging several LHS classes (originating from a `VerticalMerger`) is not allowed to inherit form a simple matched rule (originating from a `Copier`) as required in our example depicted in Fig. 4. To overcome these shortcomings again imperative code blocks have to be generated emphasizing the provided abstraction of our proposed (i)MOps. Needless to say, it would be possible to transform our MOPs to different transformation languages, e.g., TGGs [12] or our own transformation language TROPIC [20], which could additionally be used to debug our mappings.

[4]http://www.eclipse.org/modeling/emf/?project=validation

6. RELATED WORK

In the following we firstly relate our approach to existing mapping tools and secondly investigate literature on transformation patterns.

Mapping Tools. In the area of MDE to the best of our knowledge only the work presented in [6] provides predefined weaving operators which roughly correspond to our MOps. Similar to our prototypical implementation they base on AMW and specify their operators by extending the basic weaving model. Nevertheless, they do not provide operators dealing with heterogeneities caused by different metamodeling constructs including inheritance. In the area of data engineering, current mapping tools, e.g., Clio/Clip [16] and Mapforce[5], do not only support mappings of relational schemas but also of XML schemas. Although in XML schema type derivation, i.e., inheritance between complex types, is allowed, neither Clio/Clip nor Mapforce provide support to map schemas using different inheritance hierarchies. Finally, in the area of ontology engineering, approaches like MAFRA [15] provide mapping support for integrating different ontologies. Within the MAFRA framework a mapping language called *Semantic Bridge Ontology* (SBO) provides different means of linking concepts from a source ontology to a target ontology. In addition, SBO allows the specification of relationships between mappings whereby the `subBridgeOf` relationship is similar to the inheritance mechanism between our MOps. Different to our approach is that no pre-defined, coarse-grained MOps are available that resolve typical heterogeneities caused by inheritance, e.g., the resolution of the example in [15] requires the user to assemble several bridges which can be resolved by the $\mathtt{A_2I}$ composite MOp in our approach. Summarizing, current mapping tools provide only very limited support for mapping inheritance hierarchies.

Transformation Patterns. Concerning reusable transformation patterns, in the area of model engineering only language-dependent patterns, i.e., idioms, have been proposed so far such as patterns for graph transformations [1] and patterns for QVT [9]. The situation is different in the area of data engineering where a huge body of literature exists, that analyzes potentially occurring mapping situations irrespective of any concrete specification language like [3] and [13] to mention just a few. Besides these classifications of mapping situations, also collections of concrete patterns have been published in the area of data engineering [5] and in the area of ontology engineering [17]. Only the latter two consider patterns resolving inheritance heterogeneities to a certain extent. In [5], additionally the I_2A scenario is discussed but no comprehensive overview of all scenarios is given. Finally, a first benchmark for mapping systems [2] has been proposed recently, describing recurring mapping patterns in the form of a collection of problems that a dedicated mapping tool should be able to cope with, but again this work does not focus on heterogeneities caused by inheritance. When considering related work from more widely related areas like object-oriented refactoring and object-relational transformations, then interesting patterns (e.g., [7] and [11]) with respect to inheritance can be found. Summarizing, this rich body of existing literature just emphasizes the need for a library of reusable MOps encapsulating these recurring transformation patterns.

[5]http://www.altova.com/mapforce.html

7. DISCUSSION AND FUTURE WORK

In the course of this work, the ActivityDiagram case study of this year's Transformation Tool Contest has been conducted. This case study has been solved by the usage of 9 composite MOps encapsulating 34 kernel MOps, 85 % thereof responsible for copying, 6 % for merging and 9 % for generating. The high percentage of copying results from the fact that the case studies represents an evolution scenario. Nevertheless, one can see that by analyzing the kernel MOps one may draw conclusions about the complexity of a certain model transformation. By further critically reflecting our MOps several issues for future work have been identified.

Introduction of new Composite MOps. Most of our MOps have been explored in integration scenarios between CASE tools and UML tools. For further evaluating the proposed approach we plan to conduct several case studies comprising the complete mapping of UML 1.4 class diagrams to UML 2.2 class diagrams. Thereby, we would like to analyze how often certain patterns wrt. our current set of MOps occur. Moreover, we expect to detect recurring situations neglected so far, being candidates for new composite MOps.

Validation of Mappings. The provided kernel and composite MOps (including iMOps) are specified by means of a MM, allowing to validate a specified mapping. In order to enhance this basic validation support, enhanced analysis of the LHS MM and RHS MM should be incorporated in the future by using OCL constraints. By this we would like to ensure that only executable model transformations are derived from the mappings, i.e., errors should be detected already in the specification phase and not during runtime.

Enabling Interactive Guidance. Currently the specification of a mapping is entirely up to the transformation designer without any guidance. To alleviate this shortcoming interactive guidance should be provided supporting (i) the specification of the whole mapping (guidance in the large) and (ii) the specification of the MOps (guidance in the small). Concerning the first issue, we envision to propose strategies how to divide and conquer a whole mapping problem, e.g., abstract classes first, top down the inheritance hierarchy or containment references first. Since there are no best practices available in this context further research is needed. Concerning the second issue, wizards could be employed in the application of a MOp to reduce overhead and to minimize the number of errors, e.g., when applying the $\mathtt{A_2I}$ iMOp the transformation designer could be guided in providing correct conditions for the `Copiers`.

Debugging of Mappings. The operational semantics of our MOps is currently defined using a HOT to ATL. Thus, there is an impedance mismatch between the abstract mapping specification and the executable code, which hinders understandability and debugging on the mapping level when executing a derived transformation. Therefore, the translation to other transformation languages should be investigated, trying to identify which transformation language fits best to the mapping specification. In this respect, the applicability of our own transformation language TROPIC [20] should be investigated as well.

8. REFERENCES

[1] A. Agrawal, A. Vizhanyo, Z. Kalmar, F. Shi, A. Narayanan, and G. Karsai. Reusable idioms and patterns in graph transformation languages. *Electronic Notes in Theoretical Computer Science*, 127(1):181–192, 2005.

[2] B. Alexe, W.-C. Tan, and Y. Velegrakis. STBenchmark: Towards a Benchmark for Mapping Systems. *VLDB Endowment*, 1(1):230–244, 2008.

[3] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[4] J. Bézivin. On the Unification Power of Models. *Journal on SoSyM*, 4(2):171–188, 2005.

[5] M. Blaha and W. Premerlani. A catalog of object model transformations. In *Proc. of WCRE'96*, 1996.

[6] M. Del Fabro and P. Valduriez. Towards the efficient development of model transformations using model weaving and matching transformations. *Journal on SoSyM*, 8(3):305–324, July 2009.

[7] M. Fowler. *Refactoring - Improving the Design of Existing Code.* Addison-Wesley, 1999.

[8] R. Hull and R. King. Semantic Database Modeling: Survey, Applications, and Research Issues. *ACM Computing Surveys*, 19(3):201–260, 1987.

[9] M.-E. Iacob, M. W. A. Steen, and L. Heerink. Reusable model transformation patterns. In *Proc. of EDOCW'08*, 2008.

[10] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: A Model Transformation Tool. *Science of Computer Programming*, 72(1-2):31–39, June 2008.

[11] W. Keller. Mapping objects to tables: A pattern language. In *Proc. of EuroPLoP'97*, 1997.

[12] A. Koenigs. Model Transformation with TGGs. In *Proc. of Model Transformations in Practice Workshop of MoDELS'05*, Montego Bay, Jamaica, 2005.

[13] F. Legler and F. Naumann. A Classification of Schema Mappings and Analysis of Mapping Tools. In *Proc. of BTW'07*, 2007.

[14] H. Ma, W. Shao, L. Zhang, Z. Ma, and Y. Jiang. Applying OO metrics to assess UML meta-models. In *Proc. of UML'04*, 2004.

[15] A. Maedche, B. Motik, N. Silva, and R. Volz. MAFRA - A MApping FRAmework for Distributed Ontologies. In *Proc. of EKAW'02*, 2002.

[16] A. Raffio, D. Braga, S. Ceri, P. Papotti, and M. A. Hernández. Clip: a Visual Language for Explicit Schema Mappings. In *Proc. of ICDE'08*, 2008.

[17] F. Scharffe, J. de Bruijn, and D. Foxvog. Ontology Mediation Patterns Library, Version 2. Technical report, SEKT Project Deliverable, 2005.

[18] M. Tisi, F. Jouault, P. Fraternali, S. Ceri, and J. Bézivin. On the Use of Higher-Order Model Transformations. In *Proc. of ECMDA-FA'09*, 2009.

[19] M. Wimmer, G. Kappel, A. Kusel, W. Retschitzegger, J. Schönböck, and W. Schwinger. Surviving the Heterogeneity Jungle with Composite Mapping Operators. In *Proc. of ICMT'10*, 2010.

[20] M. Wimmer, A. Kusel, J. Schönböck, G. Kappel, W. Retschitzegger, and W. Schwinger. Reviving QVT Relations: Model-Based Debugging Using Colored Petri Nets. In *Proc. of MoDELS'09*, 2009.