Model Transformations for the MDA with BOTL

Frank Marschall and Peter Braun

Institut für Informatik
Lehrstuhl Prof. Dr. Manfred Broy,
Software & Systems Engineering
Boltzmannstr. 3
D-85748 Garching

Abstract

In this paper we identify some basic requirements for model transformations for the Model Driven Architecture (MDA) approach and for a model transformation language that is used to specify these transformations. Basically such a language must be precise, allow reasoning about the applicability of specifications, and allow to verify that only valid models are created. Therefore we introduce the Bidirectional Object oriented Transformation Language (BOTL) and show how this language does support the verification of the desired properties. A small running example illustrates our approach.

Key words: Model Driven Architecture, Model Transformation, Metamodel, Refinement, UML

1 Introduction

The MDA (Soley, 2000; OMG-MDA, 2001) introduced by the Object Management Group (OMG) specifies a model based software engineering approach that explicitly separates models at three abstraction layers. The most abstract model, the Computational Independent Model (CIM), deals only with concepts of the application domain. A CIM is refined into a Platform Independent Model (PIM) that contains already computational information about a system but is free from platform specific realization details. These details are part of a Platform Specific Model (PSM), which is a refinement of the PIM.

To derive one model from the other mappings between models must be specified. As already stated in Miller, Mukerji (2003) one must specify mapping

rules that specify these transformations. For the specification of mapping rules it is essential to know about the metamodel of the source and target models. This fact is also reflected in the MDA metamodel.

The Unified Modelling Language (UML) (OMG-UML, 2002) is recommended and widely accepted as a specification language for MDA models. The MDA does not specify or prescribe any language for the specification of these model transformations, but currently there is a RFP for a standard to query, create views and transform MOF models for the MDA (OMG-QVT, 2003). Obviously a transformation language for the MDA must be capable to transform object oriented models.

In practice today usually XSL transformations (Clark, 1999) are used to that transform XMI (OMG-XMI, 2002) representations of models. Unfortunately XSL documents lack an intuitive, graphical representation and have some more drawbacks. Since they operate on XMI representations writing MDA transformations is a long winded and fault-prone task. Therefore for example Peltier et al. (2001) proposes a different more abstract approach, which is based upon XSL and helps developing model transformations.

Akehurst (2000) and Gerber et al. (2002) examine different techniques for specifying model transformations. The most promising stem from the area of graph grammars (Rozenberg, 1997; Schürr, 1994). They deliver a theoretical foundation for the transformation of graphs. As a theoretically founded approach they have to be adopted to the concepts of object orientation. Therefore attributed, typed graph grammars extended by constraints may be used for a deep integration with object orientation missing up to now.

Proposals for the QVT, like (DSTC, 2003), provide a specification of transformation semantics. However, none of the existing approaches provides any techniques to ensure or prove that their application won't cause any conflicts and that generated models will be conform to their metamodel. For this purpose we introduce model transformation language BOTL for the transformation of object oriented models. Being mathematically founded BOTL allows reasoning about properties of transformations.

A complete definition of the BOTL is far out of scope of this paper and can be found in Braun, Marschall (2003). Instead we aim to give the reader an idea of the BOTL, its basic concepts, its capabilities, and how the MDA approach can profit from such a language. We show exemplary how this language can be used in the context of the MDA in Section 2. Section 3 gives an overview over techniques that allow one to decide whether the application of a transformation may fail at runtime, Section 4 introduces the concepts that are needed to decide whether generated models will conform to a given target metamodel. Finally, Section 5 summarizes the results.

2 BOTL Transformations for the MDA

As the MDA requires models to be formal, there is a *metamodel* for every kind of MDA model that defines the structure of valid models of that type. A *model transformation specification* defines how a model is derived from an existing model. Thereby the newly created model is denoted as the *target model*, while the existing one is called the *source model*. For each model there must exist a metamodel, called *source* resp. *target metamodel*.

A model transformation specification is called *applicable*, if the transformation it defines can be applied for any arbitrary source model that conforms to the source metamodel. Thereby an applicable model transformation specification must be deterministic, i.e. it must produce always the same target model for a given source model. If it holds that all created target models of a model transformation are conform to the target metamodel, then the according transformation specification is called to be *metamodel conform*. Obviously applicability and metamodel conformity are crucial properties of transformation specifications, especially when the transformations is to be performed by a tool. A prerequisite for reasoning about these properties is a precise *model transformation language* for the specification of transformations.

In the following we provide a very simple example for a model transformation of a part of a platform independent model into a platform specific CORBA model. The example originally stems from Miller, Mukerji (2001) but additional package names, OCL, and natural language constraints have been omitted to keep it more easily understandable.

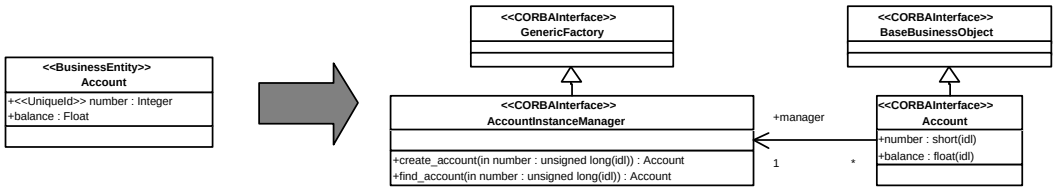

Fig. 1. A graphical representation of a PIM and a PSM that was generated from it.

The left side of Figure 1 shows the sample PIM. A CORBA specific model that was derived from it can be seen at the right side. There exists a metamodel in form of a UML profile for each of the two models. Since the profiles are easy to imagine we don't present them graphically here.

It is easy to understand the idea behind this transformation by inspecting the sample application: For every `BusinessEntity` two CORBA interfaces are created. One that inherits from the interface class `BaseBusinessObject` and an instance manager interface that inherits from the `GenericFactory` interface. Further the factory interface has a `create` and a `find` method that get the attribute stereotyped as `UniqueId` of the `BusinessEntity` as an argument.

Obviously examples are a useful way to illustrate model transformations but they are not sufficient to provide an unambiguous specification for them. Also textual specifications of transformations as they are used today are not unambiguous and allow no reasoning about the applicability of a specified mapping.

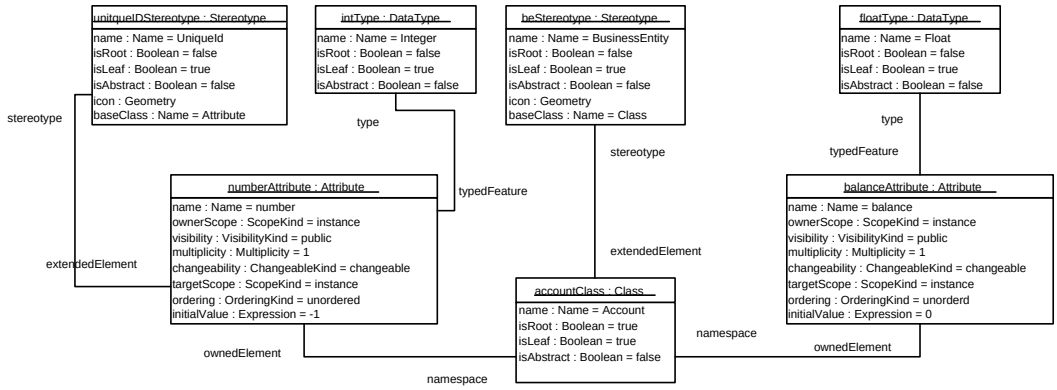

Fig. 2. The PIM from Figure 1 in terms of the UML metamodel.

Figure 2 shows the PIM as an instance of UML metamodel, while Figure 1 shows its graphical representation as an UML class diagram. The PIM contains much more elements than its representation as a class diagram but naturally all this information has to be considered by a transformation specification between our given models. Thus to precisely define a transformation between a PIM and a PSM we have to refer to model elements, because the UML metamodel level is not expressive enough therefore. E.g. one cannot differentiate between several instances of the same type in terms of the UML metamodel layer. Thus like Gogolla (2000) and Bottoni et al. (2002) we refer to instances of the UML metamodel to specify transformation rules for UML models.

We propose the rule based model transformation language BOTL for the specification MDA mappings. This language combines the illustrative clearness of a graphical specification with the precision of a formal founded language. BOTL is intended to be supported by a tool that translates fragments of a source model into target model fragments and merges the newly created fragments into a target model. BOTL transforms always object models. The source and target metamodels are consequently class models, which could be mapped to instances UML or MOF meta classes.

Figure 3 depicts a sample BOTL rule. When applied it searches in the source model (the PIM) for occurrences of a model fragment with the same structure as the pattern at its upper side and creates new fragments according to the lower pattern. The objects' identities and attribute values in the model fragment patterns contain terms. Constant values written within quotation marks, a $\diamondsuit$ indicates that the given value is irrelevant. For every match of the source pattern in the source model one new model fragment of the target model is created. The structure of this new fragment is determined by

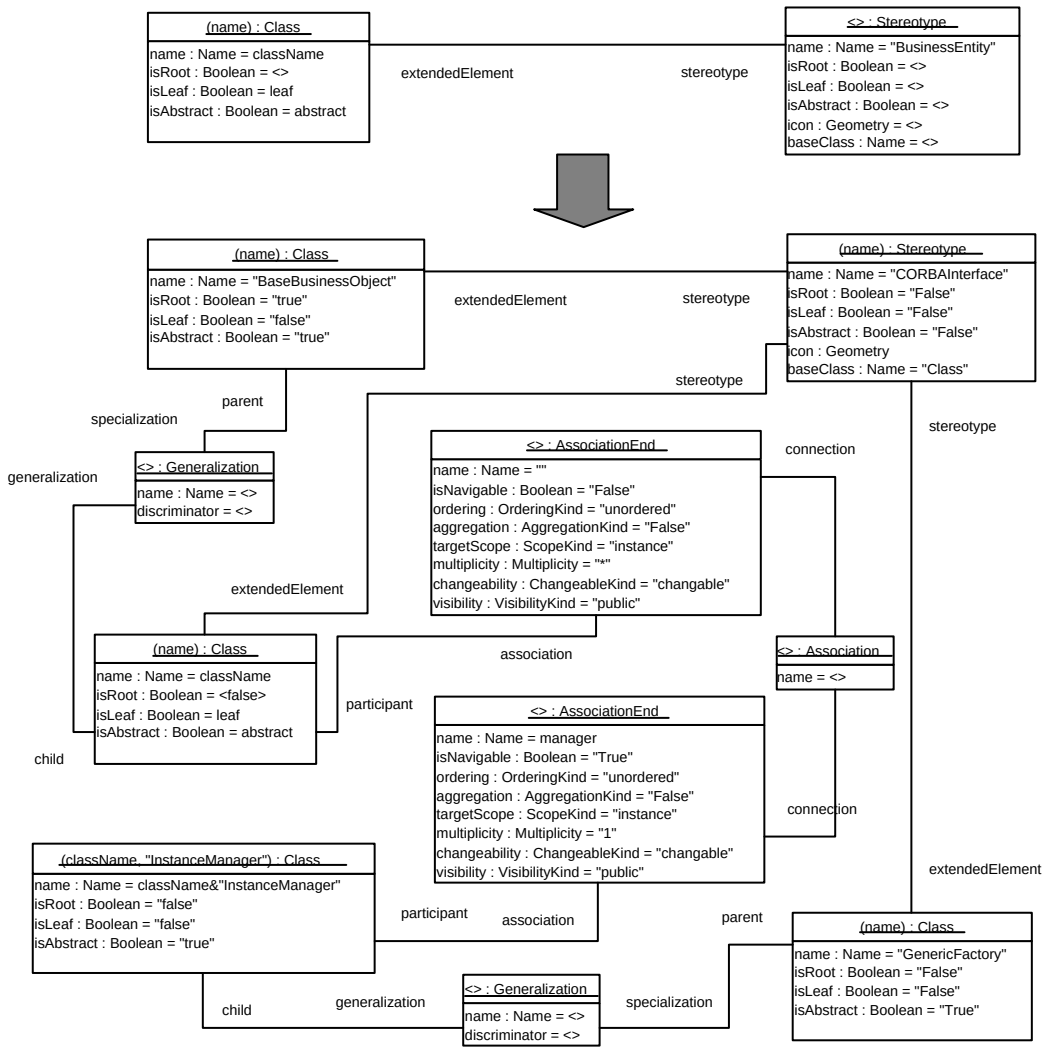

Fig. 3. A BOTL rule that specifies a part of the desired model transformation.

the target pattern. Since the model patterns serve as placeholders for existing and created model fragments we call them *source* resp. *target model variable*. The objects in a model variable are denoted as *object variables*. Regarding the QVT approach source model variables provide a mechanism to specify queries.

The new attributes' values are computed from the terms in the two model variables' attribute values. $\diamondsuit$ values in the target pattern are replaced by appropriate default values at the end of the transformation. The identity of generated objects is also determined by a term in the target object variable. If two objects with the same identity are created, then these objects are merged into one object, i.e. their attribute values are merged, whereby only $\diamondsuit$ values are overwritten with other values. Merging two associations of the same type between two objects with the same identity yields in one association that has the maximum cardinality of the two associations [1] . Figure 4 depicts the merge

[1] We regard multiple associations of the same type between objects as *one* associ-

operation exemplary. So the fragments that are created by the subsequent application of several rules are be merged to one coherent new model.

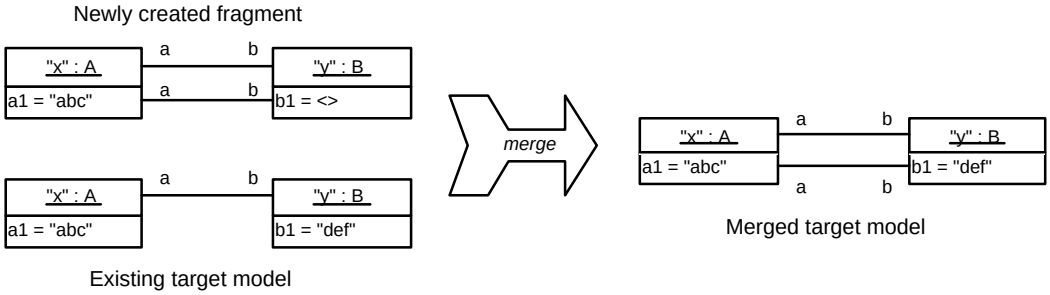

Fig. 4. Merging a newly created model fragment and an existing target model

A BOTL rule set may consist of a number of rules that all create model fragments. Thereby it is interesting to know that the order of pattern matches during one rule application as well as the order of the application of the different rules does not affect the result of a BOTL transformation. Proofs for these statements can be found in Braun, Marschall (2003).

A more detailed explanation of the application of BOTL rules is provided in Braun, Marschall (2002), a formally founded definition can be found in Braun, Marschall (2003). For this work we assume that the semantics of BOTL rules are intuitive enough to be understood by means of the given example.

The sample rule searches the source model for an object of the UML metaclass `Class` associated with a `Stereotype` with a `name` value `"BusinessEntity"`. For every found business entity the rule creates a CORBA interface and an appropriate instance manger interface that inherit from the class `GenericFactory` resp. `BaseBusinessObject` as shown in Figure 1.

The interfaces are of the type `Class`. Their identity is determined by their `name` attributes that serve as primary keys, which is indicated by the `(name)` in the identifier field of these object variables. This ensures that classes with the same name are always mapped to the same objects of the type `Class` in the PSM. The rule creates only the empty CORBA interfaces. Additional rules are needed to copy attributes into the PSM interfaces and to create appropriate `create` and `find` method declarations for the instance manager interface.

Since the instance representation of our example's PSM is even much bigger than the sample PIM we omit it here. We also skip the specification of the complete rule set for the desired transformation and instead discuss how BOTL allows reasoning about applicability and metamodel conformance, which are prerequisites for their usage in the context of the MDA.

---

ation with a given *cardinality*.

3 Applicability of BOTL specifications

For a tool supported automatic transformation of models, as e.g. the generation of a PIM from a PSM, it must be ensured that the transformation will not fail because of an inconsistent or incomplete specification. Thus a model transformation language must have a notion of applicability and allow (automatic) reasoning about it. We now introduce how BOTL supports this feature.

3.1 Applicability of BOTL rule sets

We call the property that a BOTL rule set produces output for any arbitrary source model that is conform to its source metamodel *applicability* of a rule set. Thereby the transformation of a model might fail for two different reasons.

First it might happen that it isn't possible to calculate a valid value for an attribute. This might be the case e.g. if we should calculate an attribute value from the term $\sqrt{x}$, whereby the variable x is assigned to a negative value. The second critical situation that may arise is when we try to merge two objects with the same identity that contain contradictory values for the same attributes, i.e. both values are different and differ from $\diamondsuit$.

If one of these two cases occurs, then the specified transformation cannot be applied, because there is no unambiguous solution for the resulting target model. Generally we can state that rule set is applicable, if

- each of its rules alone is applicable, and
- there are no two rules that generate two objects of the same type whereby the same attribute gets assigned a value different from $\diamondsuit$, twice.

The second postulate ensures that there are no conflicting attribute values: only one rule of the set can effectively write an attribute's value. The values of primary key attributes don't have to be considered, because two objects with different primary key attribute values won't ever have the same identity.

3.2 Applicability of a BOTL rule

There are still techniques needed to prove that a single rule is applicable. Therefore we identified three criteria that have to hold. For a formal discussion of the presented techniques we refer again to Braun, Marschall (2003):

Computable attribute values: We can determine unambiguous values for

the attributes of newly created objects.

No conflicting attributes from one object variable: An object variable in a target model variable does not create different values for the same attribute of one object during the subsequent application of a rule.

No conflicting attributes from different object variables: Different object variables of the same type don't create conflicting attribute values, too.

The following sections give a rough overview on how these properties can be verified. Again we just aim to give the reader an idea of the applied concepts.

3.3 Computable attribute values

The terms in the model variables together with the values of the source and the created target model fragment form an equational system of the kind:

$$\begin{aligned} \text{"Found attribute/id value"} &= \text{"Term in source object variable"} \\ \text{"New object's attribute/id value"} &= \text{"Term in target object variable"} \end{aligned}$$

If we regard a created objects attribute/id values as the unknown variables, then the system must have an unique solution. For the example in Figure 3 the value for the attribute `name` of a newly created instance manger is obtained from the following equational system:

$$\begin{aligned} \text{theFoundClass.}\texttt{name} &= \texttt{className} \\ \text{theNewInstanceMgr.}\texttt{name} &= \texttt{className \& "InstanceManager"} \\ \Rightarrow \text{theNewInstanceMgr.}\texttt{name} &= \text{theFoundClass.}\texttt{name\&"InstanceManager"} \end{aligned}$$

3.4 No conflicting attributes from one object

Whenever an attribute's *att* value different from $\diamondsuit$ is written into a newly created object with the identity *id* it must hold for every match that:

> The set of source objects, that are used to compute the value of *att*, is determined by the set of source objects, that are used to compute *id*.

Therefore BOTL provides some basic algorithms to determine,

- on which source object variables an identity, that is created from a given target object variable, depends,
- on which source object variables the value, that is created from a given attribute of an target object variable depends, and

- whether it holds for two sets A and B of source object variables that: Whenever the elements of A match to the same source objects, then the elements of B match always the same source objects, too.

The last item might hold, if e.g. the objects matched to B have always to-one associations to objects matched by A according to the source metamodel.
In the example from Figure 3 this statement is e.g. easy to prove for the object of type `Class` at the lower left corner. Its identifier is determined by the `Class` object of the source model variable and the only attribute value that is not constant (`name`) is also determined by the same source model variable object.

3.5 No conflicting attributes from different objects

The simplest way to ensure this property is to prohibit two or more object variables of the same type in a target model variable. But BOTL also provides another technique to prove this property, if this simple heuristic doesn't work.

The proof technique is based on an equational system that reflects the situation when two different target object variables match to the same target object during two rule applications. In this case the equational system has a restricted solution that reflects the situation when this case may occur. Now consider a set of additional equations, which state that all generated attribute values generated during these two applications are identical. If these new equations do not further restrict the systems solution the property does hold.

The latter technique can be used to prove for the sample rule that there are no conflicts that stem from different target object variables of the class `Class`.

4 Metamodel Conformance of BOTL Specifications

While applicability states that a model transformation specification is realizable, metamodel conformance states that a transformation will yield to valid target models, according to the target metamodel. This section presents the basic concepts that are used to show that a given BOTL rule set generates models that are conform to a target metamodel.

Within BOTL a model is regarded as *metamodel conform*, if

- all objects in the model are of a type that occurs in the metamodel,
- all associations in the model are of a type that occurs in the metamodel and connect objects of the correct types,

- every object in the model has not more outgoing associations of a type than the class association does allow (we denote this property as *upper bounds conformance*), and
- every object has not less outgoing associations of a type than the class association does allow (we denote this property as *lower bounds conformance*).

The first two properties are easy to verify. It has simply to be ensured that the first two postulates do hold for all target model variables. If this is the case, then the application of the rule set can't create any invalid objects or associations. These properties are already guaranteed by the syntax of BOTL rules. Note that BOTL in contrast to the UML does not yet guarantee that no sequences of composite or aggregated objects occur in a target model.

Generally it holds that a BOTL rule set is metamodel conform, if it is *applicable*, *upper bounds conform*, and *lower bounds conform*.

4.1 Upper Bounds Conformance

To verify that a rule set is upper bounds conform it has to be determined for every association in every target model variable how often this association might be maximally created as an outgoing association from the same object. This information can only be gained by reasoning about the identities of the generated objects that associations connect. Generally there are four possibilities for the identity of a new object:

(1) It results from a target object variable with a fixed identifier.
(2) It results from a target object variable with $\diamondsuit$ as its identifier term. I.e. whenever the rule is applied a new object with a unique id is created.
(3) The object's identifier depends on a set of source objects as defined in the rule. I.e. the identifier can be calculated from this set of source objects.
(4) It is not possible to make any statements about the object's identifier.

According to this we can make several estimations about the maximum number of outgoing association of an object that stems from the same model variable association. E.g. if we know that the identifiers of the connected objects variables are both constants or both $\diamondsuit$, then we know that the rule can create only one outgoing association of the given type for every created object.

A case of special interest is when a created target object's identity is one-to-one dependent on the identity of a set of matched source objects. Regarding the source metamodel and the source model variable we can reason about how often this rule can be maximally applied creating the same object. Together with the information about the identifiers at the other hand of the association we can calculate a value that states how often the association might occur as

an outgoing association from the same object.

The obtained values are summed up for every end of every association type and compared to the maximal allowed multiplicity of this association type to determine whether this multiplicity constraint might be violated or not.

Since some associations could vanish during the merge process as already indicated in Figure 4, BOTL further comes with a mechanism to determine whether an association in a model variable is redundant. In this case we don't have to consider it for the verification of upper bounds conformance.

4.2 Lower bounds conformance

There is a very simple heuristic to verify that a rule set is lower bounds conform: if every target object variable has the required minimum of outgoing associations according to the target metamodel, then the same holds for all models generated from this rule. This is, because when a new object is created all required outgoing associations are created within the same rule application.

However in some cases this heuristic might not be strong enough to prove lower bounds conformance. For these cases BOTL defines techniques that are very similar to those described in the previous section.

5 Conclusion

In this paper we first highlighted the essential role that appropriate techniques for model transformations play for the MDA approach. A human readable but unambiguous model transformation language is a prerequisite for the successful application of this approach. But such a language must also allow to reason about the applicability of specifications and their ability to guarantee that it creates only valid (i.e. metamodel conform) models.

Therefore we introduced the basic concepts and features of the Bidirectional Object oriented Transformation Language (BOTL). BOTL allows to specify transformations among object oriented models and to verify the desired properties of applicability and metamodel conformance at specification time. Since a detailed discussion of the BOTL verification techniques would have exceeded the scope of the paper, only the most important features and techniques were presented to give the reader an idea about the capabilities of the language.

We propose BOTL as a language for the specification of mappings between the different model layers of the MDA. Further BOTL could be easily extended

to specify transformations on a single model. We already use BOTL within the project KOGITO to specify views by defining a mapping of the abstract syntax of a views description technique into a common conceptual model.

Currently we are working on a tool support for BOTL that allows one to specify rules, verify their correctness and generate code for the transformation of various kinds of object oriented models.

References

David H Akehurst: Model Translation: A UML-based specification technique and active implementation approach, Phd. thesis, University of Kent, 2000

Peter Braun and Frank Marschall: Transforming Object Oriented Models with BOTL, In: Graph Transformation (ICGT) 2002, Editor: Paolo Bottoni and Mark Minas, ENCTS series 72.3, Elsevier Science B. V.

Peter Braun and Frank Marschall: BOTL—The Bidirectional Object Oriented Transformation Language, Technical Report, TU München, 2003

J. Clark: XSL Transformations (XSLT) 1.0, WWW Consortium (W3C), 1999

DSTC: Query / View / Transformations, Initial Submission, 2003

A. Gerber, M. Lawley, K. Raymond, J. Steel and A. Wood: Transformation: The Missing Link of MDA, In: Graph Transformation (ICGT), 2002 Editor: Paolo Bottoni and Mark Minas, ENCTS series 72.3, Elsevier Science B. V.

Martin Gogolla: Graph Transformations on the UML Metamodel, ICLAB Workshop on Graph Transformations and Visual Modeling Techniques, 2000, Carleton Scientific

P. Bottoni, F. Parisi-Presicce, G. Taentzter: Coordinated Distributed Diagram Transformation for Software Evolution, Software Evolution through Transformation, ENTCS 74, 2002

Model Driven Architecture (MDA), Editors: J. Miller and J. Mukerji, 2001

MDA Guide Version 1.0, Editors: Joaquin Miller and Jishnu Mukerji, 2001

OMG Model Driven Architecture (MDA), 2001

OMG MOF 2.0 Query / View / Transformations, Request for Proposal, 2003

OMG Unified Modeling Language Specification (Action Semantics), 2002

OMG XML Metadata Interchange (XMI) Specification, Januar 2002

Mikaël Peltier, Jean Bézivin and Gabriel Guillaume: MTRANS: A general framework, based on XSLT, for model transformations Workshop on Transformations in UML (WTUML), Genova, Italy, 2001

Grzegorz Rozenberg: Handbook of Graph Grammars and Computing by Graph Transformation, World Scientific, 1997

Andy Schürr: Specification of Graph Translators with Triple Graph Grammars, WG'94 20th Int. Workshop on Graph-Theoretic Concepts in Computer Science, 1994, Editor: G. Tinhofer, LNCS 903, Springer

Richard Soley: Model Driven Architecture, OMG Staff Strategy Group, 2000