Literature Study on Model Transformations

Matthias Biehl
Embedded Control Systems
Royal Institute of Technology
Stockholm, Sweden
biehl@md.kth.se

July 2010

Version 20101027222900
TRITA-MMK 2010:07
ISSN 1400-1179
ISRN/KTH/MMK/R-10/07-SE

Abstract

Model transformation is a central concept in model-driven development approaches, as it provides a mechanism for automating the manipulation of models. In this document we survey and classify existing model transformation technology. The classification differentiates between the problem space, i.e. characteristics of the problem to be solved by model transformation technology, and the mechanism, i.e. characteristics of the model transformation language. We show typical usage scenarios for model transformations and identify characteristics of the problems that can be solved with the help of model transformations. We synthesize a unifying classification scheme for model transformation languages based on several existing classification schemes. We introduce a selection of model transformation tools available today and compare them using our classification scheme.

Contents

1 Introduction

In model-driven development a series of models is created, refined and maintained. Models are the primary artifacts of the development and contain information that supports the various stages of the development process. Models can capture information of different lifecycle stages and development activities, such as requirements, design, implementation, testing, quality analysis, simulation, verification. In addition, models can represent different views of the system and represent the system on different levels of abstraction.

Model transformations provide a mechanism for automatically creating or updating target models based on information contained in existing source models, e.g. the creation of code from a design model or the translation of a UML Class Diagram into an Entity Relationship Diagram. The automatic creation of models through model transformations provides a mechanism for the systematic reuse of information. Assuming that the model transformation is correctly defined, it can be used to ensure consistency between different models.

1.1 Related Fields

The theoretical foundations of model transformation are rewrite systems and semi-Thue systems [16] from theoretical computer science. The practical foundations of model transformation techniques are laid by compiler construction [1]. Compilers translate programs written in a higher programming language (such as C++) to assembly language or representations that are close to the executable instructions of the processor. Model transformations solve a wider range of problems, including integration, analysis and simulation. Thus model transformation techniques are more universally applicable than a compiler. A compiler can even be viewed as a special application of a model transformation. Many of the techniques used in model transformation have a corresponding technique in compiler construction, with a slightly different name. For example the metamodel of model transformation corresponds to a grammar in compiler construction. Compilers typically have a pipeline architecture of lexical analysis, syntax analysis, semantic analysis, intermediate code generation and code generation. The same steps are performed by a model transformation engine.

There is a high level of automation in compiler construction. There are several specific tools to support each phase of compiler construction, such as a parser generator, scanner generator and code generator. An example is the special-purpose language Lex/YACC that is used for create parsers. Compiler generators create a compiler based on a specification and are comparable to a higher-order model transformation.

The research field of *program transformation* is related to model transformation, as it also uses techniques from compiler construction [58]. Program transformations are usually applied for analysis or optimization of certain quality attributes. Program transformations are typically unidirectional and are based on mathematically oriented concepts such as term rewriting, functional programming and attributed grammars. *Refactoring* is a well known program transformation technique used to improve the maintainability of source code without changing its behavior [23].

Generative programming is a software engineering paradigm based on describing software system families. Given a particular requirements specification,

a highly customized and optimized product can be automatically manufactured on demand from elementary, reusable implementation components by means of configuration knowledge [20]. Generative programming is closely related to model-driven development.

1.2 Overview of this Document

This document is structured as follows. In chapter 2 we define model transformation and introduce the terminology we use in this document. In chapter 3 we describe typical scenarios for the application of model transformations and provide examples. We proceed to identify the significant characteristics of problems that can be solved by model transformation and classify them in chapter 4. In chapter 5 we identify the properties of model transformation languages. These properties can be used to classify existing approaches and discuss a specific subsets of approaches. We introduce some state-of-the-art tools for model transformation in chapter 6 and use the classification scheme developed in chapter 5 to compare them. We conclude this report in chapter 7 with a discussion of the advantages and disadvantages of model transformations.

2 Terminology

2.1 Definition of Model Transformation

Model transformation is a young field and there are several competing, yet partly overlapping definitions of the terms.

Tratt [55] defines model transformation very widely as *"a program that mutates one model into another"*. The Object Management Group (OMG), an industry association for standardization within software engineering, defines model transformation in the context of the model-driven architecture (MDA) as *"the process of converting a model into another model of the same system"* [42]. Kleppe et al. [32] define model transformation as the *"automatic generation of a target model from a source model, according to a transformation description"*.

Mens et al. [39] extend this definition by also allowing several models as input or output and define model transformation as the *"automatic generation of one or multiple target models from one or multiple source models, according to a transformation description"*. This is the definition we will use in this document. Figure 1 illustrates the components of a model transformation.

2.2 Auxiliary Terminology

Model: A model is a simplified representation of a system that helps to gain a better understanding of the system [50, 7]. Models are often expressed in dedicated domain-specific languages or general purpose modeling languages such as UML [49]. Models are often represented graphically, but do not need to be.

Metamodel: A metamodel of a model X describes the structure that model X must follow to be valid. A metamodel can be compared to a grammar in language design. Precisely defined metamodels are a prerequisite for

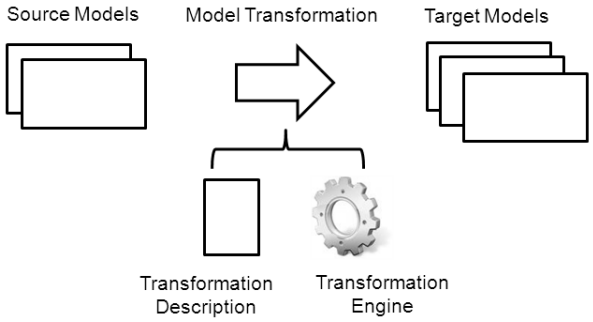

Figure 1: The components of a model transformation.

model transformations [40]. Figure 2 illustrates the metalevels of a model transformation.

Metametamodel: A metametamodel of model X is the metamodel used to describe the metamodel of model X [22]. It can be compared to the grammar of the language that is used to describe the grammar of the language X (e.g. EBNF). Standards and well established frameworks for metametamodels exist, such as MOF or Ecore. MOF is a standard for defining metamodels that is defined by the OMG [45]. MOF comes in two versions: Complete MOF (CMOF) and Essential MOF (EMOF). A commonly used implementation of EMOF is Ecore, defined by the Eclipse Modeling Framework (EMF) [52].

Model Transformation Paradigm/Approach: A model transformation paradigm or approach is the design principle on which the model transformation language is built, e.g. imperative, operational, functional, declarative or relational.

Model Transformation Language: A model transformation language is a vocabulary and a grammar with well-defined semantics for performing model transformations. The language is based on a certain *model transformation paradigm*.

Model Transformation Description: A model transformation description expresses how one or more source models are transformed into one or more target models. It is written in a *model transformation language*. If the language of a transformation description is rule-based, the transformation description is a set of *model transformation rules* [33]. A model transformation description is sometimes also called *model transformation definition*, *model transformation code* or *model transformation program*.

Model Transformation Rule: A model transformation rule in a description is the smallest entity within a model transformation. It describes how a fragment of the source model can be transformed into a fragment of the target model [33]. A rule contains a source pattern and a target pattern. For each occurrence of the source pattern in the source model, a target pattern is created in the target model. In the context of graph

transformation the source pattern is also called the *left-hand side (LHS)* and the target pattern is called *right hand-side (RHS)*.

Model Transformation Engine/Tool: A model transformation engine or tool executes or interprets the *model transformation description*. It applies the *model transformation description* on the *source model* to produce the *target model*. Transformation engines are also called *rewrite engines*. Examples for model transformation engines are SmartQVT (cf. section 6.4) or ATL (cf. section 6.2). For executing a model transformation an engine/tool typically needs to perform the following steps [55].

- identify elements in the source model that need to be transformed
- for each of the identified elements produce the associated target elements
- produce tracing information that links the source and target elements affected by this rule

Source Model: In the context of a model transformation, a model can take on the role of a source model, if it is an input to the transformation. The source model conforms to the source metamodel. One or more source models are the input of a model transformation.

Target Model: In the context of a model transformation, a model can take on the role of a target model, if it is an output of the transformation. The target model conforms to the target metamodel. A model transformation can have one or more target models. The term target model is only used for transformations that are model-to-model transformations.

Transformation Models: The transformation description can be represented using a model [6]. This allows the transformation model to be the source model or target model of another model transformation. Figure 2 illustrates a transformation model and its relation to elements on other metalevels.

Higher-order Transformation (HOT): A model transformation description having a transformation model as source or target model [54].

Model Driven Architecture (MDA): Standard [44] for Model-driven development by the Object Management Group (OMG), where *Platform Independent Models* (see below) are transformed to *Platform Specific Models* (see below).

Platform Independent Model (PIM): A PIM is a part of the standard for the *Model-driven Architecture* (MDA). PIM is a role that a model can take, where the model does not contain information about the platform used. Note that the term platform is not very concise. Note also that a model can be independent of one platform and dependent on another platform at the same time [44].

Platform Specific Model (PSM): A PSM is a part of the standard for the *Model-driven Architecture* (MDA). PSM is a role that a model can take, where the model does contain information about the platform used.

Technical Space: The term technical space or technological space is the technology used to represent models [37, 8]. The technology includes file formats, data structures, parsers, and facilities to manipulate the data. Examples for technical spaces are XML (Extensible Markup Language) [13], XMI (XML for Metadata Interchange) [46] or EMF (Eclipse Modeling Framework) [52].

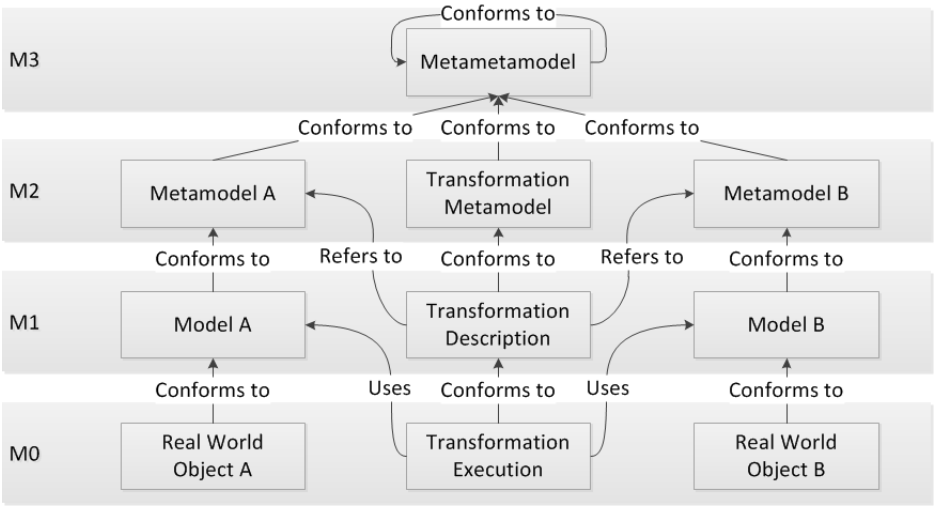

Figure 2: The metalevels of a model transformation.

3 Typical Uses of Model Transformation

Model transformations can be used for various tasks in model-driven development, e.g. for modifying, creating, adapting, merging, weaving or filtering models. The reuse of information captured in models is common to all of these tasks. Instead of creating artifacts from scratch, model transformations enable the use of information that was once captured as a model and build on it. In this chapter we will look at some specific use cases for model transformation throughout the development process.

Model transformations can be used in different phases of the development process. The V-model development process [56] is depicted in figure 3. The left leg of the V depicts the process of creating a solution by incremental refinement. This refinement starts with the requirements elicited from the user, proceeds with the development of a concept and design until an implementation is created. The right leg of the V depicts verification and validation.

In the following sections we study how model transformations can be used to support the different development tasks in the V-model. In figure 3 the use of model transformations is depicted by an arrow.

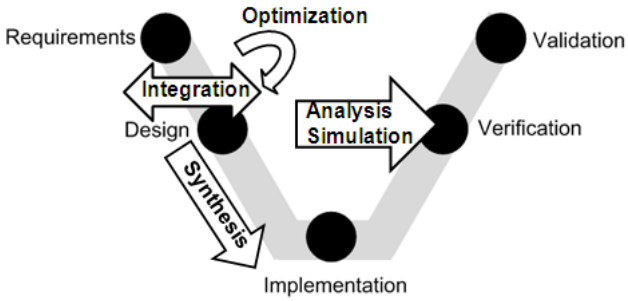

Figure 3: The V-model development process depicted as a grey V, the artifacts produced during this process are depicted by black circles. The use of model transformation the different development tasks is depicted by an arrow.

3.1 Synthesis

A transformation for synthesis is a refinement that adds details to the model. Source and target models can have the same metamodel or different ones. The level of detail is increased and it usually involves moving from a higher abstraction level to a lower one. In figure 3 synthesis can cover any stretch on the left leg of the V-cycle. Usually there is a chain of several model-to-model transformations and a final model-to-code transformation. Code generation is a special case of a synthesis transformation, where source code is produced from models of a higher level of abstraction. A synthesis transformation can refine a model in several ways: decomposition of concepts of a higher level, choice of algorithm, specialization of abstractions for a certain usage context and concretization [20].

3.2 Integration

3.2.1 Tool Integration

A number of different development tools are available to create, manipulate, analyze and simulate models. The models created are usually tool-specific, i.e. each tool expects a model that corresponds to a different metamodel and uses different technical spaces for representation. This is the reason why it is not always possible for theses tools to exchange data, hampering tool interoperability. Tool integration technology bridges the semantic and syntactic gaps between different metamodels and their tools. In a tool integration context, model transformations can be used to translate between two different metamodels or to keep two models synchronized and consistent. Model transformations can be used to establish a mapping between the metamodels of the different tools, allowing a translation between valid instances of the metamodels.

3.2.2 Model Merging

Information from two or more models might need to be merged into a common model. In the simplest case, all involved models correspond to the same metamodel. Even merging models corresponding to different metamodels is possible. Model merging is used in aspect oriented modeling [24] or model weaving [5].

3.3 Analysis, Simulation and Optimization

Model transformations can support simple and more complex analysis activities. Simple analysis can be performed by the transformation, such as model metric calculation. An example is the computation of a metric for similarity [17], which uses a model transformation written in ATL. More complex analysis or simulations can be supported by model transformation as well, by tool integration of external analysis technology. In the latter approach, input models for external analysis tools are created, as shown for example in [9, 10].

Optimization transformations are usually endogenous transformations, which focus on improving one or several non-functional properties. For example optimizations performed by a compiler are usually targeted towards performance enhancements. The optimization of models can focus on different properties such as evolvability, dependability or modularity. A refactoring is an example of an optimization transformation; it improves maintainability, evolvability and understandability.

4 Classification Scheme for Model Transformation Problems

In earlier research mainly tools and languages for model transformation have been classified and evaluated. Model transformation languages have been classified by Czarnecki et al. [15] and by Mens et al. [39]. Different model transformation tools have been evaluated by Huber [27]. The study concluded that no model transformation tool is absolutely better than another one. Instead a suitable model transformation tool exists for almost every kind of model transformation problem. This is why we propose a classification scheme for model transformation problems.

In this document we define the term *model transformation problem* as a problem that we would like to solve using a model transformation. A model transformation problem is a specific instance of the typical scenarios for model transformation described in section 3.

In the following we identify the characteristic properties of model transformation problems forming a classification. Such a classification can be helpful when deciding which model transformation language and engine is suited for solving the problem. This is useful as a large number of different alternative model transformation languages and engines is available. We have identified the following properties of model transformation problems:

- Change of Abstraction
- Change of Metamodels
- Supported Technical Spaces
- Supported Number of Models
- Supported Target Type
- Preservation of Properties

4.1 Change of Abstraction

Model transformations can change the level of abstraction between source and target model. The level of abstraction is a measure of the amount of details in a model. Model transformations either introduce new detail, reduce the amount of detail or leave it unchanged. This property is independent of the change in the metamodel (c.f. section 4.2); source and target metamodel can be the same or different.

- A *vertical transformation* changes the level of abstraction. The level of abstraction can be increased by a refinement transformation or it can be decreased by an abstraction transformation.
  - A *refinement transformation* produces the target model by adding details to the source model. A change in the metamodel might be necessary for this step. In the MDA, this type of model transformation is used to transform a platform independent model into a platform specific model.
  - An *abstraction transformation* produces the target model by reducing the amount of detail.
- A *horizontal transformation* changes the representation of the model and does not change the level of abstraction of the model. Examples are pretty-printing, improving the graphical layout, refactoring, and translating the model to a similar metamodel. Translation transformations produce the target model by expressing the same information found in the source model in a different metamodel, where the degree of detail remains the same.

4.2 Change of Metamodels

We can differentiate whether source and target metamodels are the same or different [39].

- The source and target metamodels of *endogenous transformations* are the same. When producing the target model, the transformation usually also changes only a specific part of the source model, to the largest extent source and target model are the same. This type of transformation is also called *rephrasing transformation* [57].
- *Exogenous transformations* map concepts between different metamodels. This type of transformation is also called *translation transformation* [57].

4.3 Supported Technical Spaces

Models are represented using different *technical spaces* [8]. The technical space of the models limits the set of transformation engines that can be used. Crossing the boundaries of technical spaces is not supported by all transformation tools.

4.4 Supported Number of Models

A model transformation can have several source models and several target models [39].

- The minimum number of models involved is one, where source and target model are the same. In this case the target model is created by modifying specific parts in the existing source model. The transformation assumes that source and target model are identical, except for the parts mentioned in the transformation description. The transformation is called *in-place transformation.*

- Most transformations involve two models, with distinct source and target models. The transformation assumes that the target model is empty and if it contains information, the information is overwritten. After execution of the transformation, the target model only contains information that is explicitly generated.

- A transformation can involve several source models and combine the information found in them into the target model. A transformation can also produce several target models, which often - but not necessarily - reference each other.

4.5 Supported Target Type

We can distinguish model transformations with respect to the type of the target. The target can be model or text.

- *Model-to-model* transformations create elements of the target model. Elements in the source model are mapped to elements in the target model.

- *Model-to-text* transformations create arbitrary text. Elements in the source model map to arbitrary fragment of text. Since the text lacks structure, model-to-text transformations are much harder to analyze. If the text produced by the transformation is source code, the transformation is also called *model-to-code* transformation or code transformation.

4.6 Preservation of Properties

Transformations can be built in such a way that source model and target model have a common property, which is not changed by the transformation [39].

4.6.1 Semantics-preserving

If the source and target metamodels are similar, a mapping can be found that is semantics preserving, i.e. the meaning of the two models is the same, even though it is represented in a different technical space or using a different abstract syntax.

Semantics-preserving program transformations are defined as changing the way computations are performed without changing the values computed [61]. Examples of semantics-preserving transformations are performance improvements or refactorings. In both cases the transformation does not change the outcome of the computation, but improves quality attributes. In the case of performance improvements the required resources and the time to output are reduced, thereby improving execution time and performance. Refactoring is the process of changing the internal structure of a model without simultaneously

changing the externally observable behavior or functionality of the corresponding program. Refactoring transformations are semantics-preserving, since they do not change the meaning of the model, but improve the structure and quality of the model.

If source and target metamodels contain fundamentally different assumptions, it might be difficult to completely preserve the semantics. A model transformation can approximate a preservation of the meaning of the model. Approximation can preserve the essential properties of the model. An example is the mapping from a complex state machine to a simplified state machine or the mapping between a nonlinear mathematical function and a linear mathematical function.

4.6.2 Behavior-preserving

A transformation is behavior-preserving if the explicit or implicit constraints of the behavior in the source model remain fulfilled in the target model after the transformation has been executed. An example is a transformation from a model to code, where the code produces output values that are slightly different from those predicted by the corresponding simulation of the model. Even though the transformation from model to code is not semantics-preserving, it is behavior-preserving.

4.6.3 Syntax-preserving

A syntax preserving transformation is usually an endogenous horizontal transformation that does not change the abstract syntax of the model. An example is a transformation for improving the graphical layout which preserves the abstract syntax but changes the concrete syntax, e.g. by placing graphical model elements in a new layout on the drawing canvas.

5 Classification Scheme for Model Transformation Languages

In this chapter we introduce the characteristics and properties of model transformation languages. The purpose of this listing is to create a classification framework for exploring and evaluating how model transformation languages differ from each other. Together with the problem classification introduced in section 4, this can be used for selecting a tool with appropriate properties for the task.

The framework proposed in this chapter is based on the classification by Czarnecki et al. [15] and on the taxonomy by Mens et al. [39]. Mens has applied his classification on graph transformations [41] and Koch has used the classification for studying transformation technology for web engineering [34].

5.1 Paradigm

Model transformation languages follow different language paradigms. We introduce the different paradigms in the following sections.

5.1.1 Imperative/Operational

Imperative languages specify a sequential control flow and provide means to describe how the transformation language is supposed to be executed. The constructs and language concepts of imperative model transformation languages are similar to those of general purpose programming languages such as Java or C/C++. The languages offer a high level of control to the programmer. This provides flexibility and allows for efficient implementations. The transformation is described as a sequence of actions, which is especially useful if the order of a set of transformation rules needs to be controlled explicitly [39].

5.1.2 Declarative/Relational

Declarative languages do not offer explicit control flow. Instead of *how* the transformation should be executed, the focus is on *what* should be mapped by the transformation. Declarative model transformations describe the relationship between the source and the target metamodels and this relationship can be interpreted bidirectional. For a declarative transformation between the models A and B, the transformation description can be executed in both directions: A $\mapsto$ B and B $\mapsto$ A. Graph transformation is a subcategory of declarative languages, see section 5.1.4. Declarative languages are in general compact and transformation descriptions are generally short and concise.

5.1.3 Hybrid

Hybrid transformation languages offer both imperative language constructs and declarative language constructs. They leave it up to the user whether to use imperative or declarative language constructs.

5.1.4 Graph Transformation

Graph transformation languages build on theoretical foundations of algebraic graph grammars and are a subcategory of declarative languages. Graph transformations have interesting theoretical properties and are often used in formal approaches and proofs. Models are interpreted as graphs, and graph transformations manipulate subgraphs.

Triple Graph Grammars (TGG) are a way of describing graph transformations. They have rules that are specified by three graphs:

- Left-hand side graph: subgraph of the source graph
- Right-hand side graph: subgraph of the target graph
- Correspondence graph: describes the mapping between elements of the left-hand side graph and elements of the right-hand side graph.

The left-hand side describes the precondition for the application of the rule, the right-hand side describes the postcondition of the rule.

5.1.5 Template-Based

Template-based languages are used for model-to-text transformations. Templates contain fragments of the target text and a metaprogram that can access

the source model. Usually template-based languages are combined with the visitor pattern [25] to traverse the internal structure of a model.

5.1.6 Direct Manipulation

General purpose programming languages can be used to implement model transformations. Libraries to read and write model data can be used. As an advantage, programmers do not need to learn a new language. On the other side, the implementations tend to become large and unmaintainable.

5.2 Rule Application Control

Transformation languages offer different mechanisms for determining when and where a transformation rule is applied.

- *Implicit* control does not allows specifying direct control of the order of rule application.
- *Explicit* control specifies the execution order together with the rules.
- *External* control specifies the order separately from the rules.
- *Rule application scoping* restricts the transformation to affect only parts of the model. The restriction can be either on the source model or on the target model.

5.3 Rule Scheduling

Rule scheduling determines the order of rule application.

- *Rule Selection* controls when a rule is applied. Rule selection can be either deterministic (interactive, explicit, control resolution) or non-deterministic.
- *Rule Iteration* uses recursion, looping or fixpoint operation.
- *Phasing* determines that in a certain phase only certain rules can be executed.

5.4 Rule Organization

Large model transformations contain a number of rules that need to be organized. Model transformation languages offer different ways to group, compose and reuse rules. Rules can be grouped according to the source model, the target model or independently. Different techniques for modularizing rules exist, e.g. [38].

It is desirable to reuse transformations and transformation rules, due to the potential efficiency and quality improvements. One way of reuse is composition of simple rules to build more complex rules [26, 60, 14].

- Transformations can be composed *internally*. Composition needs to be supported by the transformation language.
- Transformations can be composed *externally*. Composition of transformations of different languages can be combined.

5.5 Traceability

A trace may be produced as a side effect of the transformation. It provides a log of the execution of the transformation. It maps elements of the source model that were matched by a transformation rule to the elements of the target model that were produced by the transformation rule. Tracing functionality can be built-in into the tool, or it can be implemented as part of the transformation description.

The traces can be stored in the source model, in the target model or in a separate place. Traces can be captured as separate models, i.e. trace models, corresponding to a trace metamodel. The model transformation language QVT (see section 6.3) defines such a trace metamodel. Traces can be used as a basis for synchronization or incremental execution of model transformations.

5.6 Directionality

Transformations are used for mapping source models to target models. Transformation languages allow interpreting the mapping unidirectionally or multidirectionally.

- *Unidirectional* languages allow a mapping from source to target model.
- *Multidirectional* languages allow an interpretation of the rule in several directions. The same rule can be applied from source to target and from target to source. If a single source and a single target model are used, the mapping is called bidirectional.

5.7 Incremental Model Transformation

When a source model is changed and a transformation has previously generated a corresponding target model, some model transformation engines allow incremental updates of the target model. That way, the target model does not need to be regenerated completely, but just the part affected by the change. This is especially relevant if large models are involved.

Tratt differentiates in [55] between statefull and persistent transformations.

- *Non-incremental or statefull* transitions regenerate the complete model, incremental updates are not possible.
- *Incremental or persistent* transformations on the other hand allow updates in the source model and propagate the changes to the target model.

Traces can be used as a foundation for realizing incremental model transformations. Note that in-place transformations (c.f. section 4.4) are inherently incremental. An incremental model transformation can be incremental regarding the source or the target model.

- A *target-incremental* model transformation updates the target without rebuilding complete target.
- A *source-incremental* model transformation minimizes the number of source-elements that need to be rechecked on an incremental model transformation.

Another aspect of incremental model transformation is the preservation of user edits in the target model. Manual additions made by the user in the target model are preserved even when the target is regenerated.

5.8 Representation of the Transformation

Model transformations can be represented as text or as a model. If model transformations are models [6], it is possible to use model transformations to manipulate other model transformations, so called higher-order transformations.

6 Model Transformation Languages, Tools and Standards

In this section we briefly introduce different model transformation languages and tools.

6.1 EMF Henshin

EMF Henshin [12] is a continuation of the EMF Tiger [11] transformation language. It is an in-place model-to-model transformation language using triple graph grammars (TGG). It is based on the Eclipse Modeling Framework EMF [52].

The transformation description is a transformation model consisting of a left-hand-side graph, a right-hand-side and a list of correspondence mappings. The graph nodes are model element instances of the source metamodel and the target metamodel, respectively. It is thus possible to create higher-order transformations with EMF Henshin. There is no built-in support for creating traces, no support for multi-directionality or incremental model transformation.

6.2 ATL

The ATLAS Transformation Language (ATL) [29] is a hybrid model-to-model transformation language. ATL supports both declarative and imperative constructs. The preferred style is declarative, which allows a cleaner and simpler implementation for simple mappings. However, imperative constructs are provided so that some mappings that are too complex to be handled declaratively can still be specified. An ATL transformation program is composed of rules that describe how to create and initialize the elements of the target models. The language is specified both as a metamodel and as a textual concrete syntax. ATL is integrated in the Eclipse development environment and can handle models based on EMF. ATL also provides support for models using EMF-based UML profiles.

ATL-code is compiled and then executed by the ATL transformation engine. ATL supports only unidirectional transformations. ATL offers dedicated support for tracing. The order of the rule execution is determined automatically, with the exception of lazy rules, which need to be called explicitly. Helper functions provide imperative constructs. ATL does not support incremental model transformation, so a complete source model is read and complete target model is created. Manual changes in the target model are not preserved. ATL supports

a mode for in-place transformation, called the *refining mode*. It has limitations and cannot be used in combination with certain constructs, e.g. with lazy rules.

6.3 Query/View/Transformation (QVT)

Query/View/Transformation (QVT) is a standardized language for model transformation [47, 36] established by the Object Management Group (OMG). QVT uses the Object Constraint Language (OCL) [48], Meta Object Facility (MOF) [45] and is aligned with the Model Driven Architecture (MDA) [44].

QVT defines three languages for model-to-model transformations. QVT defines both a textual concrete syntax and a XMI-based metamodel for creating model representations of QVT transformations. QVT has a blackbox mechanism that allows calling external code from within the transformation. For the tool implementation of each of the three languages QVT defines four conformance classes for interoperability: *syntax executable* (ability to execute QVT in the concrete syntax), *XMI executable* (ability to execute QVT in a serialized XMI model), *syntax exportable* (ability to export QVT into the concrete syntax), *XMI exportable* (ability to export QVT into a serialized XMI model).

QVT defines three transformation languages:

- *QVT Relational* is a high-level declarative transformation language. Both a graphical and a textual syntax are defined for QVT. The language supports the specification of bidirectional transformations. When a bidirectional transformation is executed, the execution direction needs to be specified. A transformation is specified as a set of relations between the source and target metamodel that must hold true. This transformation can be used to check two models for consistency, to enforce consistency by modifying the target model, to synchronize two models and for in-place transformations. It supports complex pattern matching using OCL. Trace models are created implicitly. The semantics is defined by a mapping to QVT Core.

- *QVT Core* is a simple, low-level declarative model transformation language. It serves as a foundation for QVT Relational and is equally expressive. It supports pattern matching over a flat set of variables, where the variables of source, target and trace models are treated symmetrically. Trace models must be defined explicitly.

- *QVT Operational* is an imperative model transformation language that extends QVT Relational with imperative constructs. The transformations are unidirectional. It uses implicit trace models.

Model transformation engines that are conform to the QVT standard include for example SmartQVT (cf. section 6.4) and ModelMorf (cf. section 6.5).

6.4 SmartQVT

SmartQVT [2] is an implementation of a transformation engine for Operational QVT. It is an imperative language for model-to-model transformation for EMF-based models. The transformation description is compiled into Java code and supports the QVT blackbox mechanism to call external code.

It offers built-in tracing support, in addition it offers reflection to access tracing information, such as the target object corresponding to a source object or the source object corresponding to a target object. There is support for control parameters and higher order rules.

Incremental model transformation is not supported, as the complete source model is read and the complete target model is created. There is currently no support for multi-directionality.

6.5 ModelMorf

ModelMorf [53] is an implementation of the OMG standard QVT Relational. It is a declarative model-to-model transformation. It supports multi-directionality, so the same rule can be used to map in both directions. It supports target-incremental model transformation (called change propagation semantics), so if the source model changes only the changed part of the model is transformed. It provides built-in functionality to create traces. It is possible to create in-place transformations. Furthermore, it is possible to compose model transformations to build and extend complex transformations from simpler ones.

There is currently no support for aspect-orientation, reflexion or source-incremental model transformation.

6.6 OpenArchitecureWare (OAW)

OAW [18] integrates a number of tools for model transformations into a coherent framework. OAW provides a workflow specification language and the transformation language Xpand. The workflow language is used to control the transformation process and to specify the sequence of transformations between the different models. The Xpand transformation language is a template-based, imperative language for model-to-text transformations. OAW is distributed as a plugin of the Eclipse platform and is able to handle EMF models (Eclipse Modeling Framework).

6.7 Kermeta

Kermeta [21, 43] is a general purpose modeling and imperative programming language, also able to perform transformations. It offers EMF-based metamodeling, constraints, checks, transformation and behavior support.

Models and metamodels need to be explicitly loaded and stored. Target elements need to be explicitly instantiated and added to the target model, which requires more code. Rule application control and rule scheduling needs to be specified explicitly by the user.

Kermeta supports reflection, exception handling and aspect-orientation. There is no built-in support for traceability and multi-directionality. Incremental model transformation is not supported, so the complete source model is read and complete target model is created when the transformation is executed.

6.8 ETL

The Epsilon Transformation Language (ETL) [35] is a hybrid model-to-model transformation language. It is part of the Epsilon model management infras-

tructure. It can handle several source and several target models. It offers rule scheduling functionality: *lazy rules* are only executed, when they are explicitly called, *guarded rules* are only executed if their guard evaluates to true, greedy rules are executed whenever possible. Rules can be reused and extended through rule inheritance. External code can be executed from within the transformation rule.

6.9 XML Stylesheet Language Transformations (XSLT)

XSLT [59] is a functional transformation language for manipulating XML data. Being a functional language, rules have to be called explicitly. There is no built-in traceability support and rules are strictly unidirectional. Transformations are stateful, so there is no support for incremental transformation. XSLT transformation descriptions are themselves XML documents, so higher-order transformations can be realized. Due to the fact the XSLT was initially developed to transform XML documents into HTML documents, XSLT is limited to simple transformations [4].

6.10 More...

Many more model transformation languages exist, we list some names and references here, without claiming to be complete: Moflon [3], mediniQVT [28], Textual Concrete Syntax (TCS) [30], XText [19], Tefkat [51], MOLA [31], MT, SiTra, MofLog, GreAT, GenGen, Beanbag, UMT, UMLX, ATOM, VIATRA, BOTL, XDE Transformations, Codagen Architect Transformations, b+m Generator Framework, OptimaJ Transformations, ArcStyler Transformations, MPS Transformation, Microsoft DSL Tool Transformations, Metaedit+ Transformations, AndroMDA, JET, FUUT-je, GMT, Jamda, Fujaba Transformations, TXL, Stratego

7 Conclusion

Model transformations can be used for different tasks throughout the development process for manipulating models. Model transformation descriptions are expressed in model transformation languages.

We clarified the terminology of model transformations and showed potential usage scenarios for model transformations. We first identified some characteristics of the problems that can be solved with the help of model transformations. We then synthesized a classification scheme for model transformations from existing classifications. Several languages, tools and a standard for model transformations have been developed in recent years. We used the classification scheme to classify model transformation languages, to compare the properties the languages can offer.

The field of model transformation is an active research field, and the latest approaches could not be covered in this report. More information can be found in relevant conference proceedings and journals: International Conference on Model Transformation (ICMT), International Conference on Model Driven Engineering Languages and Systems (MODELS), Journal for Software and Systems Modeling, Journal of Systems and Software.

Acknowledgements

This work has been partially funded by the FP7 project ATESST2 and the ARTEMIS projects CESAR and iFEST. The author would like to thank Jad El-khoury for reviewing this document.

References

[1] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*. Addison Wesley, January 1986. [Online]. Available: http://www.worldcat.org/isbn/0201100886

[2] F. Alizon, M. Belaunde, G. DuPre, B. Nicolas, S. Poivre, and J. Simonin, "Les modèles dans l'action à france télécom avec smartqvt," in *Génie logiciel: Congrès Journées Neptune No5*, 2007. [Online]. Available: http://smartqvt.elibel.tm.fr

[3] C. Amelunxen, A. Königs, T. Rötschke, and A. Schürr, "Metamodeling with MOFLON," *Applications of Graph Transformations with Industrial Relevance*, pp. 573–574, 2008. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-89020-1_40

[4] G. J. Bex, S. Maneth, and F. Neven, "A formal model for an expressive fragment of xslt," *Inf. Syst.*, vol. 27, no. 1, pp. 21–39, March 2002. [Online]. Available: http://dx.doi.org/10.1016/S0306-4379(01)00033-3

[5] J. Bézivin, S. Bouzitouna, M. Del Fabro, M. P. Gervais, F. Jouault, D. Kolovos, I. Kurtev, and R. F. Paige, "A canonical scheme for model composition," in *Model Driven Architecture Foundations and Applications*, 2006, pp. 346–360. [Online]. Available: http://dx.doi.org/10.1007/11787044_26

[6] J. Bézivin, F. Büttner, M. Gogolla, F. Jouault, I. Kurtev, and A. Lindow, "Model transformations? transformation models!" in *in Proceedings of Model Driven Engineering Languages and Systems (MODELS2006)*, 2006, pp. 440–453. [Online]. Available: http://dx.doi.org/10.1007/11880240_31

[7] J. Bézivin and O. Gerbé, "Towards a precise definition of the omg/mda framework," in *ASE '01: Proceedings of the 16th IEEE international conference on Automated software engineering*. Washington, DC, USA: IEEE Computer Society, 2001, p. 273.

[8] J. Bézivin, F. Jouault, and P. Valduriez, "On the need for megamodels," in *Proceedings of Workshop on Best Practices for Model-Driven Software Development at the 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications.*, Vancouver, British Columbia, Canada, October 2004. [Online]. Available: http://www.softmetaware.com/oopsla2004/mdsd-workshop.html

[9] M. Biehl, C. DeJiu, and M. Törngren, "Integrating safety analysis into the model-based development toolchain of automotive embedded systems," in *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES 2010)*, April 2010, pp. 125+.

[10] M. Biehl, C.-J. Sjöstedt, and M. Törngren, "A modular tool integration approach - experiences from two case studies," in *3rd Workshop on Model-Driven Tool & Process Integration (MDTPI 2010) at the European Conference on Modeling Foundations and Applications (ECMFA 2010)*, June 2010.

[11] E. Biermann, K. Ehrig, C. Köhler, G. Kuhns, G. Taentzer, and E. Weiss, "Graphical definition of in-place transformations in the Eclipse Modeling Framework," in *MODELS 2006*, vol. 4199, 2006, pp. 425–439. [Online]. Available: http://dx.doi.org/10.1007/11880240_30

[12] E. Biermann, S. Jurack, C. Krause, T. Arendt, and G. Taentzer, "Henshin: Advanced concepts and tools for in-place EMF model transformations," in *MODELS 2010*, October 2010. [Online]. Available: http://www.eclipse.org/modeling/emft/henshin

[13] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible markup language (XML) 1.0 (fifth edition)," W3C, Tech. Rep., November 2008. [Online]. Available: http://www.w3.org/TR/REC-xml

[14] J. Cuadrado and J. Molina, "Modularization of model transformations through a phasing mechanism," *Software and Systems Modeling*, 2009. [Online]. Available: http://dx.doi.org/10.1007/s10270-008-0093-0

[15] K. Czarnecki and S. Helsen, "Feature-based survey of model transformation approaches," *IBM Systems Journal*, vol. 45, no. 3, pp. 621–645, 2006.

[16] M. Davis, R. Sigal, and E. J. Weyuker, *Computability, Complexity, and Languages, Second Edition: Fundamentals of Theoretical Computer Science (Computer Science and Scientific Computing)*, 2nd ed. Morgan Kaufmann, February 1994. [Online]. Available: http://www.worldcat.org/isbn/0122063821

[17] M. D. Del Fabro and P. Valduriez, "Semi-automatic model integration using matching transformations and weaving models," in *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*. New York, NY, USA: ACM Press, 2007, pp. 963–970. [Online]. Available: http://dx.doi.org/10.1145/1244002.1244215

[18] S. Efftinge, P. Friese, A. Haase, C. Kadura, B. Kolb, D. Moroff, K. Thoms, and M. Voelter, "openarchitectureware user guide," openArchitectureWare Community, Tech. Rep., 2007.

[19] S. Efftinge and M. Völter, "oaw xtext: A framework for textual dsls," in *Eclipsecon Summit Europe 2006*, November 2006. [Online]. Available: http://www.eclipse.org/Xtext

[20] U. W. Eisenecker and K. Czarnecki, *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[21] J.-r. Falleri, M. Huchard, and C. Nebut, "Towards a traceability framework for model transformations in kermeta," in *In: ECMDA-TW Workshop*, 2006. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.99.6894

[22] J. M. Favre, "Towards a basic theory to model model driven engineering," in *In Workshop on Software Model Engineering, WISME 2004, joint event with UML2004*, 2004.

[23] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: improving the design of existing code*, ser. Object Technology Series. Addison-Wesley, 1999.

[24] R. France, I. Ray, G. Georg, and S. Ghosh, "An aspect-oriented approach to early design modeling," in *IEE Proceedings Software*, vol. 151, no. 4, 2004, pp. 173–185.

[25] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*. Addison-Wesley, 1996.

[26] S. Hidaka, Z. Hu, H. Kato, and K. Nakano, "Towards a compositional approach to model transformation for software development," in *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*. New York, NY, USA: ACM, 2009, pp. 468–475. [Online]. Available: http://dx.doi.org/http://doi.acm.org/10.1145/1529282.1529383

[27] P. Huber, "The model transformation language jungle - an evaluation and extension of existing approaches," Master's thesis, Technische Universität Wien, May 2008. [Online]. Available: http://www.big.tuwien.ac.at/teaching/theses/ma/huber.pdf

[28] IKV++ Technologies. mediniQVT. [Online]. Available: http://projects.ikv.de/qvt

[29] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, "ATL: a model transformation tool," *Science of Computer Programming*, vol. 72, pp. 31–39, June 2008.

[30] F. Jouault, J. Bézivin, and I. Kurtev, "Tcs: A dsl for the specification of textual concrete syntaxes in model engineering," in *GPCE 2006*, 2006. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.103.1929

[31] E. Kalnina, A. Kalnins, E. Celms, and A. Sostaks, "Graphical template language for transformation synthesis," in *Proceedings of Second International Conference SLE 2009*, 2009, pp. 244–253. [Online]. Available: http://mola.mii.lu.lv/

[32] A. Kleppe, J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture—Practice and Promise*. Addison-Wesley, 2003.

[33] ——, *MDA Explained: The Model Driven Architecture—Practice and Promise*. Addison-Wesley, 2003.

[34] N. Koch, "Classification of model transformation techniques used in uml-based web engineering," *Software, IET*, vol. 1, no. 3, pp. 98–111, 2007. [Online]. Available: http://dx.doi.org/10.1049/iet-sen:20060063

[35] D. Kolovos, R. Paige, and F. Polack, "The epsilon transformation language," in *Theory and Practice of Model Transformations*, ser. Lecture Notes in Computer Science, A. Vallecillo, J. Gray, and A. Pierantonio, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, vol. 5063, ch. 4, pp. 46–60. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-69927-9_4

[36] I. Kurtev, "State of the art of qvt: A model transformation language standard," *Applications of Graph Transformations*, pp. 377–393, 2008. [Online]. Available: http://www.springerlink.com/content/2g55gw5260q2740h/fulltext.pdf

[37] I. Kurtev, J. Bézivin, and M. Aksit, "Technological spaces: An initial appraisal," in *CoopIS, DOA 2002 Federated Conferences, Industrial track*, 2002. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.109.332

[38] I. Kurtev, K. van den Berg, and F. Jouault, "Rule-based modularization in model transformation languages illustrated with atl," *Sci. Comput. Program.*, vol. 68, no. 3, pp. 138–154, October 2007. [Online]. Available: http://dx.doi.org/10.1016/j.scico.2007.05.006

[39] T. Mens and P. Van Gorp, "A taxonomy of model transformation," *Electr. Notes Theor. Comput. Sci*, vol. 152, pp. 125–142, 2006.

[40] ——, "A taxonomy of model transformation," *Electr. Notes Theor. Comput. Sci*, vol. 152, pp. 125–142, 2006.

[41] T. Mens, P. Van Gorp, D. Varró, and G. Karsai, "Applying a model transformation taxonomy to graph transformation technology," in *Proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005)*, vol. 152, 2006, pp. 143–159. [Online]. Available: http://dx.doi.org/10.1016/j.entcs.2005.10.022

[42] J. Miller and J. Mukerji, "Mda guide version 1.0," Mai 2003.

[43] N. Moha, S. Sen, C. Faucher, O. Barais, and J.-M. Jézéquel, "Evaluation of kermeta for solving graph-based problems," *International Journal on Software Tools for Technology Transfer (STTT)*, April 2010. [Online]. Available: http://dx.doi.org/10.1007/s10009-010-0150-1

[44] OMG, "Model Driven Architecture (MDA) Guide," OMG, Tech. Rep., 2003. [Online]. Available: http://www.omg.org/mda/

[45] ——, "Meta Object Facility (MOF), v2.0," OMG, Tech. Rep., January 2006. [Online]. Available: http://www.omg.org/spec/MOF/2.0/

[46] ——, "MOF 2.0 / XMI Mapping Specification, v2.1.1," OMG, Tech. Rep., December 2007. [Online]. Available: http://www.omg.org/technology/documents/formal/xmi.htm

[47] ——, "MOF 2.0 Query / View / Transformation," OMG, Tech. Rep., December 2009. [Online]. Available: http://www.omg.org/spec/QVT

[48] ——, "Object Constraint Language (OCL)," OMG, Tech. Rep., 2010. [Online]. Available: http://www.omg.org/spec/OCL/2.2

[49] ——. (2010) Unified Modeling Language (UML). [Online]. Available: http://www.omg.com/uml/

[50] E. Seidewitz, "What models mean," *IEEE Softw.*, vol. 20, no. 5, pp. 26–32, 2003. [Online]. Available: http://dx.doi.org/http://dx.doi.org/10.1109/MS.2003.1231147

[51] J. Steel and M. Lawley, "Model-based test driven development of the tefkat model-transformation engine," *Software Reliability Engineering, International Symposium on*, vol. 0, pp. 151–160, 2004. [Online]. Available: http://dx.doi.org/10.1109/ISSRE.2004.23

[52] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework (2nd Edition)*, 2nd ed. Addison-Wesley Professional, January 2008. [Online]. Available: http://www.worldcat.org/isbn/0321331885

[53] Tata Consultancy Services. ModelMorf. [Online]. Available: http://121.241.184.234:8000/ModelMorf/ModelMorf.htm

[54] M. Tisi, F. Jouault, P. Fraternali, S. Ceri, and J. Bézivin, "On the use of higher-order model transformations," in *Model Driven Architecture - Foundations and Applications*, R. F. Paige, A. Hartman, and A. Rensink, Eds., vol. 5562. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 18–33. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02674-4_3

[55] L. Tratt, "Model transformations and tool integration," *Software and Systems Modeling*, vol. 4, no. 2, pp. 112–122, May 2005. [Online]. Available: http://dx.doi.org/10.1007/s10270-004-0070-1

[56] VDI, "Design methodology for mechatronic systems (VDI 2206)," VDI, Tech. Rep., 2004.

[57] E. Visser, "A survey of rewriting strategies in program transformation systems," in *1st International Workshop on Reduction Strategies in Rewriting and Programming*, November 2001.

[58] ——, "A survey of strategies in program transformation systems," *Electronic Notes in Theoretical Computer Science*, vol. 57, 2001. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.28.4289

[59] W3c, "Xsl transformations (xslt)," W3C, Tech. Rep., November 1999. [Online]. Available: http://www.w3.org/TR/xslt

[60] D. Wagelaar, "Composition techniques for rule-based model transformation languages," *Theory and Practice of Model Transformations*, pp. 152–167, 2008. [Online]. Available: http://www.springerlink.com/content/761ru4426u037255/fulltext.pdf

[61] W. Yang, S. Horwitz, and T. Reps, "A program integration algorithm that accommodates semantics-preserving transformations," *ACM Trans. Softw. Eng. Methodol.*, vol. 1, no. 3, pp. 310–354, 1992. [Online]. Available: http://dx.doi.org/10.1145/131736.131756