CTIT Technical Report TR–CTIT–03–27, University of Twente

Model Driven Architecture: Foundations and Applications

http://trese.cs.utwente.nl/mdafa2003

Arend Rensink (Editor)

26–27 June 2003, University of Twente

A Synthesis-Based Approach to Transformations in an MDA Software Development Process

Ivan Kurtev, Klaas van den Berg

Software Engineering Group, Department of Computer Science, University of Twente
P.O. Box 217, 7500 AE, Enschede, the Netherlands
Email: {kurtev, vdberg}@cs.utwente.nl

Abstract
In an MDA software development process, models are repeatedly transformed to other models to finally achieve a set of models with enough detail to implement the system. Generally, there are multiple ways to transform one model into another model. Alternative target models differ in quality properties and the selection of a particular model is determined on the base of specific requirements. Current transformation languages only provide means to specify transformations but do not help to identify and select among alternative transformations. In this paper we propose a synthesis-based software development process with a set of techniques for constructing a transformation space for a given transformation problem. The process takes a source model, its meta-model and the meta-model of the target, and quality requirements as input and generates a transformation space.

Key words: MDA, Software Development Process, Synthesis, Model Transformations

1. Introduction

The Model Driven Architecture (MDA) approach proposed by OMG [4] aims to promote interoperability and portability. Current software development processes such as the Unified Process [1] have to be adapted to this approach. A software development process can be described using the Software Process Engineering Meta-model (SPEM) [7]. A software development process is defined as collaboration between entities called process roles that perform operations called activities on concrete, tangible entities called work products. Activities consist of a number of steps. A discipline partitions activities within a process according to a common theme. In the Unified Process nine disciplines are described, e.g. Business Modeling, Requirement Management, Analysis and Design, and Implementation.

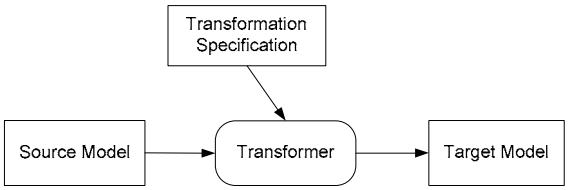

Figure 1. The generalized MDA Pattern [3]

MDA software development involves a specification of the system functionality in a set of Computation Independent Models (CIMs), Platform Independent Models (PIMs), and Platform Specific Models (PSMs). A key characteristic of the MDA is the notion of model transformation. Model transformations appears at any level: CIMs to CIMs, CIMs to PIMs, PIMs to PIMs, PIMs to PSMs and PSMs to PSMs. In the MDA Guide [3] this is captured in the MDA pattern, which can be generalized as depicted in Figure 1. The transformation specification can consists of - for example - mapping rules based on (meta-) models. A model transformation is a set of transformation rules and techniques used to operate on a source model to produce another model, the target model. In general, transformation rules relate constructs in the source model to the constructs in the target model (see Figure 2).
The source and target models are at level M1 according to the Meta Object Facility (MOF) [5] terminology and their source and target meta-models are at level M2. The source model M_A is an instance of the source meta-model MM_A and it has to be transformed to a new model that is an instance of the target meta-model MM_B. The two meta-models determine the possible transformations rules. The resulting target models may differ from each other in the qual-

ity properties they possess, such as performance or adaptability. Software engineers have to compare and choose among the alternatives based on quality requirements.

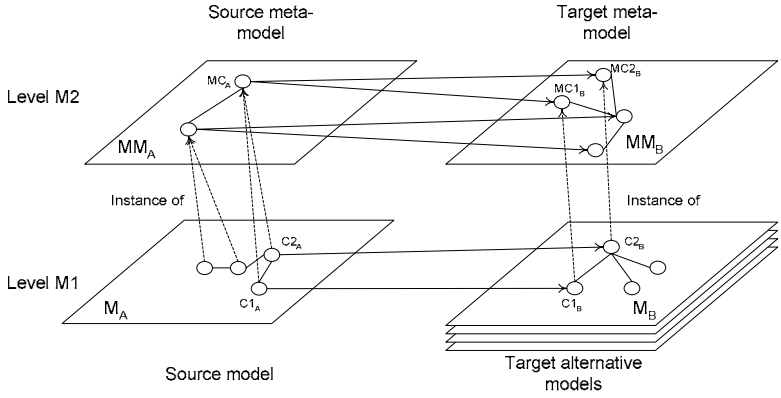

Figure 2 Multiple alternative transformations for a given source model

The diversity of quality requirements prevents the usage of a fixed set of transformation rules. For a concrete problem the software engineer must be able to identify the transformations that lead to a model with the desired quality properties. Unfortunately, current transformation languages and techniques do not provide means to identify alternative transformations and to compare them based on specific quality characteristics of the resulting models. This paper uses a synthesis-based approach to an MDA software development process. It applies a technique to explicitly model a set of alternative transformations for the source model. This technique also allows specification of quality properties of the models. The quality properties are used as selection criteria among the alternatives.

In section 2 we explain the problem addressed in the paper. Section 3 explains the synthesis-based approach to software development. Section 4 describes techniques for constructing transformation spaces and selecting alternatives from them. Section 5 discusses the applicability of these techniques in the context of the model transformations in MDA.

2. Problem Statement

An MDA software development process involves transformations from models to models as depicted in the MDA pattern. We show the presence of alternative model transformations with an example of transforming UML class diagrams to XML schemas. At level M1 we have a UML model (source) and alternative XML schemas (target). At level M2 we have the UML meta-model and an XML Schema meta-model that can be derived from the XML Schema specification [8] (There is no standard XML Schema meta-model expressed according to MOF but some proposals already exist [6]). The UML class model can be considered as the PIM and the XML-Schema as the PSM. For example, the *Class* construct defined in the UML meta-model may be mapped to Element declaration or Complex type definition construct in the XML Schema meta-model. Then at level M1 there are two possibilities for mapping each class in the source model: either to an element declaration or to a complex type. These possibilities can be combined in alternative mappings that produce alternative XML schemas. We can identify some transformations that produce alternative schemas. Even for simple source models there are more than one correct target models. The first problem is to select among the alternatives. It is rarely the case that every alternative is a good solution. Usually various requirements must be fulfilled. Software engineers are often faced with this situation but usually the process of comparing alternatives is more implicit than explicit. The second problem is that there is no support for identification of alternative transformations. The transformation to the required schema may not be always trivial and obvious and then a systematic approach is required.

3. Synthesis-based Software Development

In the Introduction we described activities and work products in SPEM. MDA work products are models, transformation specifications and transformers. An MDA software development process can be seen as a sequence of applying MDA patterns, starting with the user requirements specified in a CIM and after the final transformation resulting in an operational application. The target model in one step serves as source model in the following step. These activities can be grouped into disciplines around the MDA levels.

There is no easy fit of the MDA work products into activities as defined in the Unified Process. In this paper we demonstrate how the synthesis-based approach to software development can be applied in an MDA context. The major activities in synthesis-based software development are the following: technical problem analysis, solution domain analysis, and alternative space analysis. Each activity is refined into several steps [9]. The three activities could be carried out for each transformation.

As transformations are at the core of MDA we now focus on the third activity in which we implicitly use results of the technical problem analysis and the solution domain analysis (e.g. the analysis of the XML Schema, the Schema meta-model, the UML domain class model and the UML meta-model). A complete analysis of this case study is given in [2].

The important concepts in the MDA transformation problem are depicted in Figure 3.

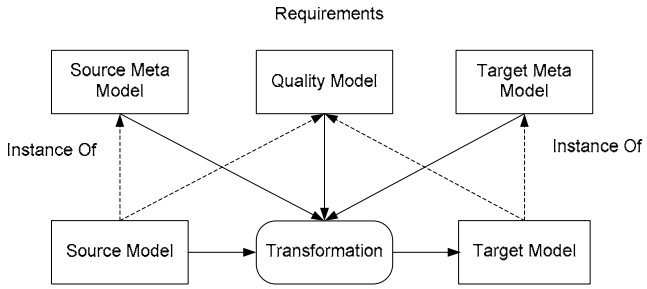

Figure 3 The transformational problem in the MDA pattern

The source model specifies the system functionality at certain level of abstraction. The source model is an instance of the source meta-model. We aim at identifying transformations that produce target model(s), which preserve the functionality in the source model and realizes that functionality on the target platform. The target meta-model (or target platform) is considered as a constraint and in some cases the required functionality cannot be implemented with the features provided by the constructs in the target meta-model. Quality requirements for the models such as adaptability, extensibility, performance, etc. are specified in the quality model. Those requirements are imposed on the source model and they must be preserved in the resulting target model where these quality properties are interpreted in terms of the target meta-model. In Figure 3 we denote the fulfillment of the quality requirements by the source and the target models as conformance to the quality model.

In summary, the transformational problem in MDA is the problem of specifying a transformer that takes the source model, its meta-model, the quality model and the target meta-model as input and generates target model(s) that conforms to the target meta-model and the quality model. The alternative space analysis is directed at the identification and specification of feasible transformations between the source model and the target model.

4. Alternative Space Analysis

In this section we describe the activity Alternative Space Analysis of the synthesis-based software development process in the context of the MDA transformation problem. The workflow for this activity is shown in Figure 4.

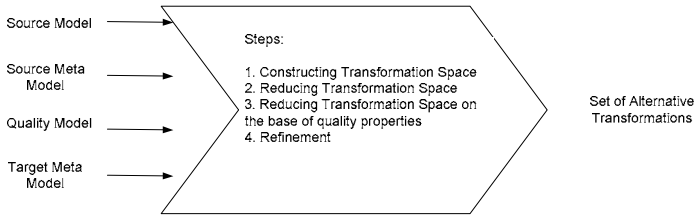

Figure 4 Workflow of the activity Alternative Space Analysis

This activity takes a source model, its meta-model, the quality model and the meta-model of target model as input and generates a *transformation space* for the source model, i.e. a set of alternative transformations for a given source model. A *transformation space* is a multidimensional space spanned over a number of independent *dimensions***.** Each dimension is associated with a number of *coordinates* that form a *coordinate set*. Every element in the space represents a transformation that produces a model that is an instance of the target meta-model. Since a transformation space may be too large some operations are defined to reduce the space. The required quality characteristics of the target model are represented in a quality model. The concepts from the quality model are woven with the source model elements and they indicate characteristics that the target model must possess. The result of the activity is a set of alternative transformations that can be used to produce the transformer required for the model transformation. We describe the process of constructing transformation space for the simple example model in Figure 5. It contains two classes related with a generalization relation. The quality requirement in that example is extensibility of the model regarding expected specializations of *Class2*.

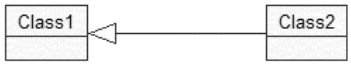

Figure 5 Example source model: UML class diagram with two classes with generalization relation

This model will be transformed to an XML schema. The example shows that even in this simple case there are multiple valid schemas that can be generated. Two alternative XML schemas for the source model are given in Figure 6.

<complexType name='class1'> …….. </complexType> <complexType name='class2'> <complexContent> <extension base='class1'> ……. </extension> </complexContent> </complexType>	<element name='c1' type='class1'/> <complexType name='class1'> …….. </complexType> <element name='c2' substitutionGroup='c1'> <complexType> <complexContent> <extension base='class1'> ……. </extension> </complexContent> </complexType> </element>
(a)	**(b)**

Figure 6 Two alternative XML schemas derived from the source model

The activity of alternative space analysis has the following 4 steps.

1. Constructing Transformation Space. The dimensions of a transformation space are determined from the constructs in the source model. A subset of the constructs in the source model is selected and one dimension is defined for each construct in that subset. A construct from the source model used to define a given dimension is always an instance of a construct from the source meta-model. The coordinate set for that dimension is determined on the base of the construct from the source meta-model. For the source meta-model construct the set of possible target constructs from the target meta-model is determined. Then this set is used to form the coordinate set for the dimension. A point in a transformation space represents an alternative transformation of the source model. For every source construct the target construct

is determined as the coordinate of the point over the dimension corresponding to that source construct.

The transformation space for the example in Figure 5 contains three dimensions: `Class1`, `Class2` and `Generalization`. The first two correspond to the classes and the third corresponds to the relation between the classes. Coordinate sets for these dimensions are defined below:

```
coordinateSet(Class1)={CT, E, MG, AG}
coordinateSet(Class2)={CT, E, MG, AG}
coordinateSet(Generalization)={Der, Subst, Cont, Ref, E}
```

Coordinate sets are derived from the constructs available in the XML Schema meta-model. Abbreviations stands for Complex Type Definition (`CT`), Element Declaration (`E`), Attribute Declaration (`A`), Attribute Group (`AG`) and Model Group Definition (`MG`). For the `Generalization` dimension we choose among the XML Schema relations. They are Derivation (`Der`), Substitution (`Subst`), Containment (`Cont`) and Reference (`Ref`). Figure 7 shows a graphical representation of the transformation space for our example. It has 4 * 4 * 5 = 80 alternatives.

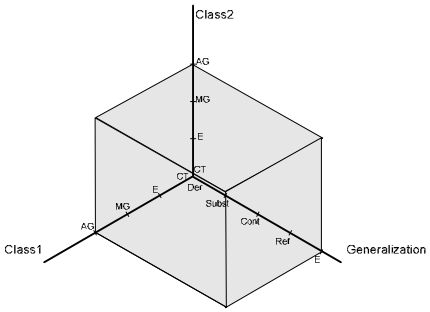

Figure 7 Transformation Space with three dimensions corresponding to two classes and generalization

2. Reducing Transformation Spaces. It is possible to generate all the alternatives in a transformation space and to compare them. The number of alternatives is usually large even for simple source models. However, it is unnecessary to generate the whole space of alternatives. Instead, the software engineer may reduce the space either by selecting or by excluding alternatives from the transformation space.

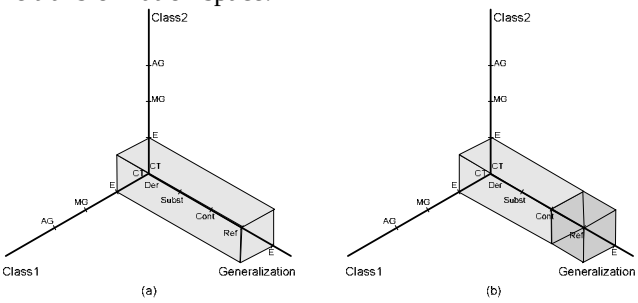

Figure 8 Transformation Spaces after selection (a) and exclusion (b) operations

For example the software engineer may decide to *select* only Element Declaration (`E`) and Complex Type (`CT`) as alternatives for UML Class. He specifies criteria for selection from the transformation space, e.g. based on best practices or heuristic rules. The space after this selection is shown in Figure 8a. Furthermore, the software engineer may decide to *exclude* Reference (`Ref`) and Element Declaration as alternatives for the generalization. The points that satisfy the criteria are excluded from the transformation space. The space after the application of the exclusion operation is shown in Figure 8b. The dark shaded area shows the part that is excluded.

3. Reducing Transformation Space on the base of Quality Properties. Transformation space may be further reduced by considering the quality requirements that the target model

must fulfill. The quality requirements are used to create the quality model. Once the software engineer has the quality model constructs explicitly represented he can identify instances of those constructs in the source model. Then, he can use selection criteria based on them.
For instance, in our example we may require extensible XML schemas that allow adding specializations of *Class2* without changing the generated schema. The quality model in that case distinguishes two types of components: extensible and inextensible. We consider *Class2* as extensible. The interpretation of the extensibility in terms of the target XML schema is that the content model of the extensible component is reused by the components created from the expected specializations of *Class2*. According to that interpretation the schema in Figure 6b is not extensible because the complex type is defined as anonymous and therefore its content cannot be reused through the schema derivation mechanism. The schema in Figure 6a allows derivations and can be considered as extensible. The process of assigning quality properties to the source model and the operations that support it are described in more details in [2].
4. Refinement. This is the last step of the activity for alternative space analysis. Once the space is sufficiently reduced the software engineer may generate the alternatives explicitly. However, these alternatives are not complete transformations yet. Some additional details are required before the transformation is specified and executed. For instance, the XML element declaration components always require a type to be defined.

5. Conclusion

We presented a synthesis-based approach for an MDA software development process with focus on the activity of alternative space analysis. In current software development processes such as the Unified Process, no explicit attention is given to exploration and selection of alternatives. For MDA there are inherently many possible alternative transformations from source to target models. In the synthesis-based approach to the software development process alternative space analysis is treated as a separate activity with specific techniques for generating and reducing the transformation space. Although transformation spaces tend to be large even for simple models, they are purely conceptual. The software engineer does not need to generate all the alternative transformations from a transformation space. A number of reduction steps are applied and the size of the space is reduced. The structure of a transformation space specified by dimensions and coordinate sets provides a framework to reason about the alternatives in general instead of per alternative individually.
Since software engineers generally have to fulfill both functional and quality requirements, they should be able to identify and compare the quality properties of the functionally equivalent alternative target models for the same source model. The proposed techniques capture the quality requirements in a quality model that can be used in criteria for reducing the transformation space.

References

1. Kruchten, P. (2000). *The Rational Unified Process*, Addison-Wesley-Longman.
2. Kurtev, I, Berg, K.G. van den, Aksit, M. (2003). *UML to XML-Schema Transformation: a Case Study in Managing Alternative Model Transformations in MDA.* To appear in FDL'03 .
3. MDA Guide (2003). Version 1.0, Object Management Group, omg/2003-05-01.
4. Miller, J., Mukerji, J. (2001). *Model Driven Architecture (MDA)*. OMG Document available at http://www.omg.org
5. OMG/MOF. (2000). *Meta Object Facility (MOF) Specification.* OMG Document available at http://www.omg.org
6. OMG/XMI. (2001). *XML Metadata Interchange (XMI) Version 2*. OMG Document available at http://www.omg.org
7. Software Process Engineering Metamodel Specification (2002). Version 1.0, Object Management Group formal/02-11-14.
8. Thompson, H., Beech, D., Maloney, M., Mendelsohn, N. (2001). *XML Schema Part 1: Structures*, W3C Recommendation. http://www.w3.org/TR/xmlschema-1
9. Tekinerdogan, B., Aksit, M. (2002). *Synthesis-Based Software Architecture Design*. In Aksit, M. (ed.) 'Software Architectures and Component Technologies', Kluwer Academic Publishers