UML to XML-Schema Transformation: a Case Study in Managing Alternative Model Transformations in MDA

Ivan Kurtev, Klaas van den Berg, and Mehmet Aksit

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

Abstract

In a Model Driven Architecture (MDA) software development process, models are repeatedly transformed to other models in order to finally achieve a set of models with enough details to implement a system. Generally, there are multiple ways to transform one model into another model. Alternative target models differ in their quality properties and the selection of a particular model is determined on the basis of specific requirements. Software engineers must be able to identify, compare and select the appropriate transformations within the given set of requirements.

The current transformation languages used for describing and executing model transformations only provide means to specify the transformations but do not help to identify and select from the alternative transformations.

In this paper we propose a process and a set of techniques for constructing a transformation space for a given transformation problem. The process uses a source model, its meta-model and the meta-model of the target as input and generates a transformation space. Every element in that space represents a transformation that produces a result that is an instance of the target meta-model. The requirements that must be fulfilled by the result are captured and represented in a quality model.

We explain our approach using an illustrative example for transforming a platform independent model expressed in UML into platform specific models that represent XML schemas. A particular quality model of extensibility is presented in the paper.

1 Introduction

The core idea of the Model Driven Architecture (MDA) approach proposed by OMG [MM01] is to specify system functionality in a set of Platform Independent Models (PIMs), separately from the specification of the implementation of that functionality on a specific platform in Platform Specific Models (PSMs). According to the MDA approach, the software development process is driven by models organized within a four layers meta-modeling stack [OMG00] where the highest layer is fixed and known as Meta Object Facility (MOF), a self-defined meta-meta model.

A key characteristic of the MDA is the notion of model transformations. A model transformation is a set of transformation rules and techniques operating on a source model to produce another model, the target model. In general, transformation rules relate the constructs in the source model to the constructs in the target model (see Figure 1).

In Figure 1, the source and target models are situated in level M1 according to the MOF terminology and their source and target meta-models are in level M2. The source model M_A is an instance of the source meta-model MM_A and 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.

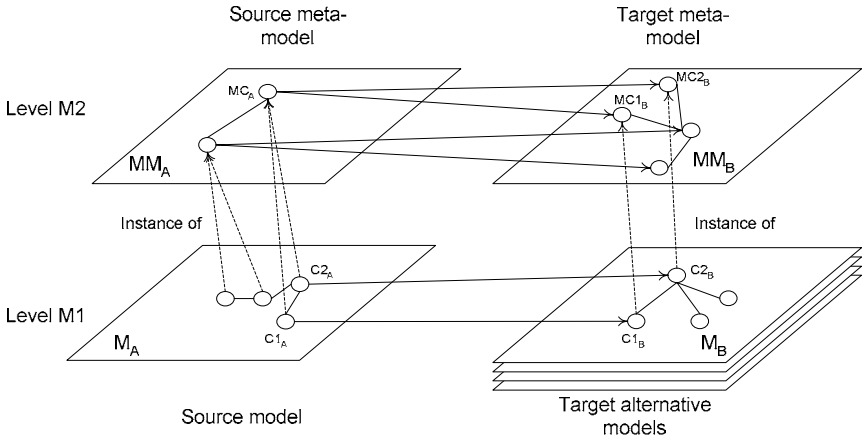

Figure 1: Multiple alternative transformations for a given source model.

Generally, for each construct of the source meta-model there are multiple constructs of the target meta-model to which it can be mapped. Assume that the construct MC_A of the source meta-model MM_A can be mapped to either of the two constructs $MC1_B$ and $MC2_B$ of the target meta-model MM_B. This introduces alternative mappings for each instance of MC_A in the source model M_A. The figure shows one possible mapping in which the instance $C1_A$ is mapped to an instance of $MC1_B$ and the instance $C2_A$ is mapped to an instance of $MC2_B$. Other combinations are generally possible and this results in multiple target models.

The resulting target models may differ from each other in the quality properties they possess. For example, assume that the target model is implemented using an object-oriented language. Generally, various implementation alternatives exist. One implementation may entail inline code and therefore display a better performance than the implementation, which clearly separates code into distinct runtime objects. The latter implementation, however, may offer more adaptability. Software engineers have to compare and choose among the alternatives using the quality requirements. 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, the current transformation languages and techniques do not provide the means to identify alternative transformations and to compare them regarding specific quality characteristics of the resulting models.

This paper uses a technique called Design Algebra to explicitly model a set of alternative transformations for the source model. This technique also allows the specification of quality properties of the target model. The quality properties are used as selection criteria among the alternatives. The approach described above is used to manage the transformations of domain models in UML into XML-Schema's.

This paper is organized as follows. Section 2 presents the example and explains the problems addressed in this paper. Section 3 describes the techniques for constructing transformation spaces and selecting alternatives from them. Section 4 discusses the applicability of these techniques in the context of the model transformations in MDA. Section 5 gives an overview of the related work and section 6 gives the conclusions.

2 Problem Statement

2.1 Transformation from a UML class model into XML schemas

We show the presence of alternative model transformations by an example of transforming UML class diagrams into XML schemas. Figure 2 depicts the concepts in this case and is a concretion of

the general picture depicted in Figure 1. The UML class model can be considered as the platform independent model and the XML-Schema as the platform specific model.

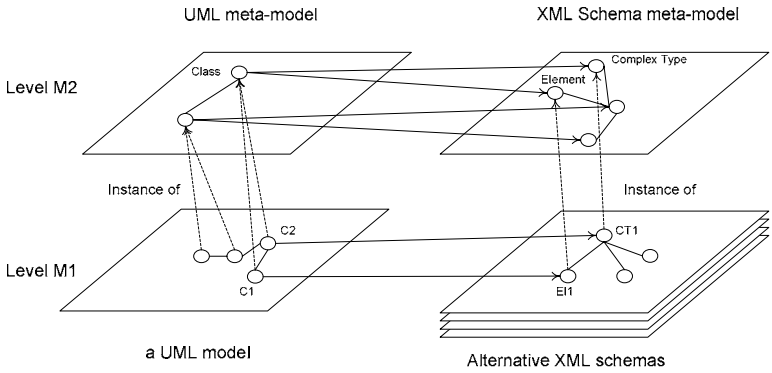

Figure 2: Multiple alternative transformations for a given UML class model into XML schemas.

In Figure 2 the source model is an UML class model and its meta-model is the UML meta-model [OMG01a]. The target meta-model is a XML Schema meta-model that can be derived from the XML Schema specification [TBM01]. The *Class* construct defined in the UML meta-model may be mapped, for example, to the 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 to alternative mappings that produce alternative XML schemas.

As a case study for this transformation problem we consider a system that supports teachers in preparing examinations. Examinations consist of questionnaires answered by students. Figure 3 shows an UML model of the questionnaires:

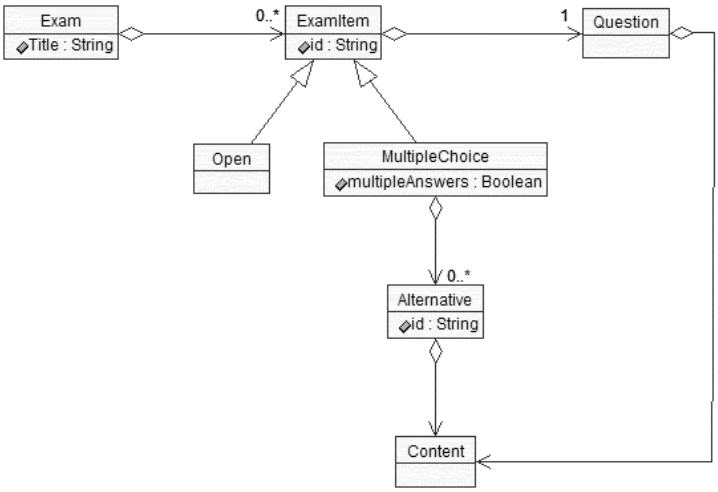

Figure 3: Example source model: UML class diagram of examination questionnaires.

Class *Exam* is used to represent examinations and contains zero or more exam items. Each exam item has exactly one question. There are two types of exam items: open and multiple-choice. In the open type, a student may give any answer. The multiple-choice type requires a student to select from a list of alternative answers. Class *Content* is used to describe the format of the questions and answers and can be expressed by any combination of text, image, audio, etc.

This model can be implemented using different techniques. As an example, we assume that the examination documents are stored as XML documents and are validated by an XML schema. The schema is derived by transforming the model shown in Figure 3 to a target model that is an instance

of the XML Schema meta-model. The actual schema expressed in XML syntax is derived from the target model.

In our example, since new exam item types may be introduced as specializations of class *ExamItem*, the schema should be able to change in the future to incorporate these new exam items. Therefore, we aim at *extensible* schemas that will remain intact even if new items are introduced.

Section 2.2 describes the problems encountered in transforming UML models to XML schemas.

2.2 Problems in transforming a UML Class Model into XML Schemas

In order to transform the UML model in Figure 3 to an XML schema, we need to map every construct in the model, i.e. all classes, attributes and relationships to appropriate constructs available in the XML Schema meta-model. There is no standard XML Schema meta-model expressed according to MOF but some proposals already exist [OMG01b]. Also, the abstract schema data model from [TBM01] may be used. We can identify some transformations that produce alternative schemas. Fragments from three schemas are shown in Figure 4. They only show how classes *ExamItem*, *Open* and *MultipleChoice* are transformed.

```
<element  name='examItem'>
  <complexType>
    <sequence>
      <choice>
        <element name='open' type='…'/>
        <element name='mutipleChoice' type='…'/>
      </choice>
      <element name='question' type='…'/>
    </sequence>
  </complexType>
</element>
```

(a)

```
<element name='examItem'
            type='examItemType'/>

<complexType name='examItemType'>
   ……..
</complexType>

<complexType name='openType'>
  <complexContent>
     <extension base='examItemType'>
      …….
     </extension>
  </complexContent>
</complexType>

<complexType name='multipleChoiceType'>
  <complexContent>
     <extension base='examItemType'>
      …….
     </extension>
  </complexContent>
</complexType>
```

(b)

```
<element name='examItem'
            type='examItemType'
            abstract='true'/>

<complexType name='examItemType'>
   ……..
</complexType>

<complexType name='openType'>
  <complexContent>
     <extension base='examItemType'>
      …….
     </extension>
  </complexContent>
</complexType>

<complexType name='multipleChoiceType'>
  <complexContent>
     <extension base='examItemType'>
      …….
     </extension>
  </complexContent>
</complexType>

<element name='open'
            type='openType'
            substitutionGroup='examItem'/>

<element name='multipleChoice'
            type='multipleChoiceType'
            substitutionGroup='examItem'/>
```

(c)

Figure 4: Three alternative schemas derived from the exam model.

In alternative (a) classes *ExamItem*, *Open* and *MultipleChoice* are transformed into complex types and elements of these types (the complex types for *Open* and *MultipleChoice* are not shown). The generalization relation between class *ExamItem* and its subclasses is mapped to containment between the corresponding elements. In alternative (b) *ExamItem* class is mapped to an element and a complex type while the two specialized classes *Open* and *MultipleChoice* are mapped to complex types derived by extension from the complex type of exam item. Alternative (c) is similar to alternative (b) but elements are declared for every class and they are organized in a substitution group.

Figure 4 shows that even for simple source models transformed intuitively several correct target models exist. The first problem is to select from the alternatives. It is rarely the case that every alternative is a good solution. Usually some requirements must be fulfilled; in our example, the schemas must be extensible. Software engineers are often faced with such a problem but usually the process of comparing among the alternatives is more implicit than explicit.

The second problem is that there is no support for the 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 Constructing and Utilizing Transformation Spaces

In this section we describe a process and techniques for constructing and utilizing a set of alternative transformations for a given source model. The process uses a source model, its meta-model and the meta-model of the target models 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 built with concepts similar to the concepts found in Design Algebra and used to build design spaces. In the context of Design Algebra a *design space* is a set of alternatives for a given design problem.)

A *transformation space* is a multidimensional space spanning 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 that help 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 combined with the source model elements and indicate certain characteristics the target model must possess. In this paper we focus on the extensibility model as a quality model. We illustrate the process by applying it to the example from section 2.

The process of constructing and utilizing a transformation space has 4 steps:

Step1. *Constructing Transformation Space.* In this step a transformation space is constructed on a set of dimensions and a coordinate set for each dimension. The process of identifying dimensions and coordinate sets is described in section 3.1.

Step 2. *Reducing Transformation Space.* Since a transformation space may be too large, operations for selection and exclusion are defined to reduce the space. These are explained in section 3.2.

Step 3. *Reducing Transformation Space on the basis of the Extensibility Model*. In this step the software engineer expresses the quality properties of the result as a quality model, in this paper the model of extensibility. The software engineer decides on the extensibility properties of the constructs from the source model and combines this information with the transformation space. This makes him aware of the desired characteristics of the target model he aims at. Extensibility properties are used to further reduce the transformation space. This step is explained in section 3.3.

Step 4. *Refinement.* Once the space has sufficiently been reduced, the alternatives can be generated. They do not represent a complete transformation and some additional tuning is required. The result of this step contains enough information for the specification of the transformation in a given language. This is described in section 3.4.

3.1 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. Some constructs of the source model do not introduce dimensions. They may be skipped if the transformation alternatives for them are considered as unimportant for the target model.

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 basis of the construct from the source meta-model. The set of possible target constructs from the target meta-model is determined for the source meta-model construct. Then this set is used to create 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.

We define two functions that will be used in this paper: `dimensions(S)` and `coordinateSet(dimension, S)`. The first function returns the set of dimensions for a given space S and the second one returns the coordinate set for a given dimension in a given space S.

A point in a transformation space S is formally represented as a tuple with components for every dimension and the coordinate in that dimension:

$$(d1.c_{d1},\ d2.c_{d2},\ldots,\ dn.c_{dn}) \quad (1)$$

where `di` is the name of the dimension and c_{di} is the coordinate of the point in that dimension. We have $c_{di} \in$ `coordinateSet(di, S)` for i=1,…,n and space S.

We will describe the process of constructing transformation space $S_{ExamSchema}$ for the example model in Figure 3. We decide not to consider transformation alternatives for the attributes of the classes. Also, for the sake of brevity only classes *Exam*, *ExamItem*, *Open* and *MultipleChoice* are considered together with the relations among them. This simplification does not affect the illustration of the basic ideas behind the formalism we are describing.

For each class and relation we define one dimension:

$$\begin{aligned}&dimensions(S_{ExamSchema})=(Exam,\ ExamItem,\ Open,\ MultipleChoice,\\ &\quad Exam\_ExamItem,\ ExamItem\_MultiChoice,\ ExamItem\_Question)\end{aligned} \quad (2)$$

In (2) `Exam`, `ExamItem`, `Open`, and `MultipleChoice` are dimensions derived from the classes with the same names, and `Exam_ExamItem`, `ExamItem_Open` and `ExamItem_MultiChoice` are dimensions derived from the relations that connect the classes used to form the name of the dimension.

In our example the classes in the source model are instances of *Class* construct defined in the UML meta-model [OMG01a] and the relations are instances of *Generalization* and *Association* constructs respectively. Coordinate sets for the dimensions are determined on the basis of the target constructs defined in the target meta-model, in this example - the XML Schema meta-model.

Despite the fact that there is no standard meta-model for XML Schemas, a set of constructs may be identified using the W3C Schema specification [TBM01]. We will use an XML Schema meta-model defined with a set of components and a set of relations among them:

$$XMLSchemaMetaModel = (C, R) \quad (3)$$

where `C` is a set of components and `R` is a set of relations. The set `C` has the following elements:

$$C = \{CT,\ ST,\ E,\ A,\ AG,\ MG\} \quad (4)$$

The elements of `C` correspond to the XML Schema abstract data model components Complex Type Definition (`CT`), Simple Type Definition (`ST`), Element Declaration (`E`), Attribute Declaration (`A`), Attribute Group (`AG`) and Model Group Definition (`MG`). Complex Type and Element Declaration have a Boolean property *abstract*. It indicates whether the type or element is abstract. The other components defined in the XML Schema abstract data model such as particles, wildcards, and identity constraints are not included here.

The set of relations `R` has the following elements:

$$R = \{Der,\ Subst,\ Cont,\ Ref\} \quad (5)$$

All relations are binary. Derivation (`Der`) corresponds either to extension or to restriction mechanisms over schema types as defined in the specifications. Substitution (`Subst`) relates two elements and corresponds to the element substitution mechanism. Containment (`Cont`) denotes participation of one component in the content model of another. Element-subelement relations and the way elements own their attributes are examples of containment. Reference (`Ref`) relation is

based on the usage of elements or attributes of types ID and IDREF(S) in the content model of the related components.

Apart from the components and relations a set of constraints may be defined that restricts the relations allowed between two components. Constraints are shown in Table 1. The columns and rows represent components. The table cells indicate the relations allowed between two of components.

From\To	**CT**	**ST**	**E**	**A**	**MG**	**AG**
CT	Der, Cont, Ref	Der, Cont	Der, Cont, Ref	Der, Cont	Cont, Ref	Cont, Ref
ST		Der	Der	Der		
E	Der, Cont, Ref	Der, Cont	Der, Cont, Ref, Subst	Der, Cont	Cont, Ref	Cont, Ref
A		Der	Der	Der		
MG	Cont, Ref	Cont	Cont, Ref		Cont, Ref	Ref
AG	Ref	Cont	Ref	Cont	Ref	Cont, Ref

Table 1: XML Schema Meta-model constraints on the relations between components[1].

Now we will identify the coordinate sets for the dimensions defined in (2). Assume that the software engineer decides that *Class* construct defined in the UML meta-model is mapped to one of the components defined in (4). Simple type (ST) can be excluded because it represents simple values. The same reasoning leads to the exclusion of attribute declaration (A) as a possible target. The remaining 4 components form a coordinate set that can be attached to the dimensions associated with the classes in the source model:

$$\begin{aligned}&\text{coordinateSet}(\text{Exam}, S_{\text{ExamSchema}})=\{\text{CT, E, MG, AG}\}\\&\text{coordinateSet}(\text{ExamItem}, S_{\text{ExamSchema}})=\{\text{CT, E, MG, AG}\}\\&\text{coordinateSet}(\text{Open}, S_{\text{ExamSchema}})=\{\text{CT, E, MG, AG}\}\\&\text{coordinateSet}(\text{MultipleChoice}, S_{\text{ExamSchema}})=\{\text{CT, E, MG, AG}\}\end{aligned} \quad (6)$$

Assume now that the software engineer chooses the set R of XML Schema relations as a coordinate set for the dimensions defined for the relations in the source model. Also, XLink standard [DMO01] uses elements to encode relations and therefore element declaration is also considered as a possible coordinate. The following three coordinate sets are defined for the dimensions corresponding to the relations in the source model:

$$\begin{aligned}&\text{coordinateSet}(\text{Exam\_ExamItem}, S_{\text{ExamSchema}})=\{\text{Der, Subst, Cont, Ref, E}\}\\&\text{coordinateSet}(\text{ExamItem\_Open}, S_{\text{ExamSchema}})=\{\text{Der, Subst, Cont, Ref, E}\}\\&\text{coordinateSet}(\text{ExamItem\_MultiChoice}, S_{\text{ExamSchema}})=\{\text{Der,Subst,Cont,Ref,E}\}\end{aligned} \quad (7)$$

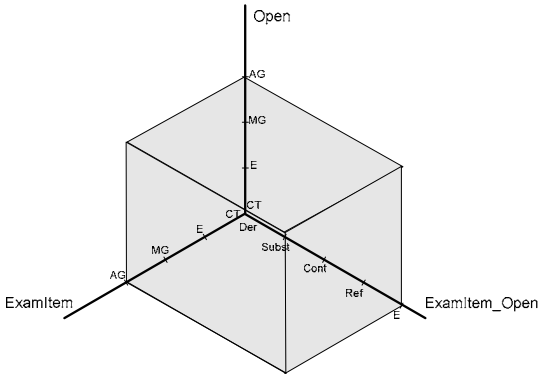

Figure 5: Transformation Space with three dimensions corresponding to classes *ExamItem* and *Open* and the generalization relation between them.

[1] Abbreviations: A=Attribute Declaration, AG=Attribute Group, Cont=Containment, CT=Complex Type Definition, Der=Derivation, E=Element Declaration, MG=Model Group Definition, Ref=Reference, ST=Simple Type Definition, Subst=Substitution

Each point in the transformation space $S_{ExamSchema}$ represents a transformation from the UML model in Figure 3 to an XML schema. An additional requirement is that the transformation must produce a schema that satisfies the constraints from Table 1.

Figure 5 shows a graphical representation of a part of the space $S_{ExamSchema}$ where only the dimensions for classes *Open*, *ExamItem* and the relation between them *ExamItem_Open* are shown together with the coordinates defined in (6) and (7).

3.2 Reducing Transformation Spaces

It is possible to generate all the alternatives in a transformation space and to compare them. However, the number of alternatives is usually large. The number of alternatives for a space S with n dimensions is calculated with the formula:

$$numAlternatives(S)=numCoordinates(D_1) *\ldots* numCoordinates(D_n) \quad (8)$$

Here `numAlternatives` and `numCoordinates` are functions defined over a space and a dimension from that space respectively. D_i denotes a dimension in the space S. `numAlternatives` returns the number of alternatives in a space and `numCoordinates` returns the number of coordinates for a given dimension.

The application of the formula (8) to the transformation space $S_{ExamSchema}$ gives 4 * 4 * 4 * 4 * 5 * 5 * 5= 32000 theoretically possible alternatives which is a large number, considering the fact that this is a simple source model of only 7 elements. Therefore it is unfeasible 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.

Two operations for selection and exclusion from a space are defined:

```
Select from S where <condition>                    (9)
Exclude from S where <condition>                   (10)
```

Operation `Select` selects from a given space S only the alternatives that satisfy the given condition, whereas the operation `Exclude` excludes from the space S the alternatives that satisfy the condition.

In our example the software engineer may decide that the classes in the source model are to be mapped either to an element declaration (E) or to a complex type definition (CT). This can be specified as a selection from the transformation space $S_{ExamSchema}$:

```
S_ReducedExamSchema1 = Select from S_ExamSchema where
                 <(ExamItem.E or ExamItem.CT) and
                 (Exam.E or Exam.CT) and
                 (Open.E or Open.CT) and
                 (MultipleChoice.E or MultipleChoice.CT)>      (11)
```

The space may be further reduced by excluding some alternatives for the relations. Assume that the software engineer decides to exclude the coordinates `Ref` and `E` for the dimensions that represent relations in the source model:

```
S_ReducedExamSchema2 = Exclude from S_ReducedExamSchema1 where
                 <(ExamItem_Open.Ref or ExamItem_Open.E) and
                 (ExamItem_MultiChoice.Ref or ExamItem_MultiChoice.E)
                  and (Exam_ExamItem.Ref or Exam_ExamItem.E)>   (12)
```

After these operations the transformation space $S_{ReducedExamSchema2}$ contains 2 * 2 * 2 * 2 * 3 * 3 * 3 = 432 alternatives.

Figure 6a shows the space defined in Figure 5 after the selection operation and Figure 6b shows the same space after the exclusion operation. The dark shaded area shows the part that is excluded.

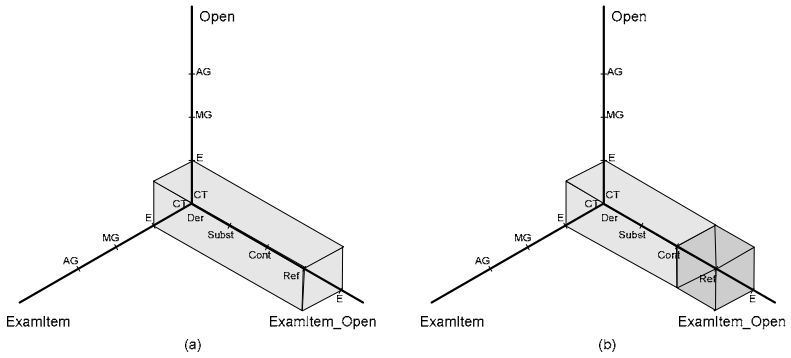

Figure 6: Transformation Spaces after selection and exclusion operations.

3.3 Reducing Transformation Space on the basis of Quality Properties

The transformation space may be further reduced by considering the quality requirements that the target model must fulfill. In our example we aim at extensible schemas that preserve the content model of some schema components if new components are added as a result of extension of the source model. This indicates some quality properties that must be satisfied by the constructs in the target model.

Quality properties are derived from a quality model. In our example, this is a model of extensibility. This model classifies constructs as either extensible or inextensible. This classification is used to form a coordinate set with two coordinates: (Exts, InExts). Here Exts indicates that a construct is extensible and InExts indicates that there is no explicit requirement for extensibility. A construct is extensible if the addition of certain components to the model is possible and does not cause changes in the construct.

We define a new transformation space $S_{ExtensibleExam}$ with the same set of dimensions as $S_{ReducedExamSchema2}$ and coordinate set (Exts, InExts) for each dimension. Assume that the software engineer decides to classify classes *ExamItem* and *Exam* as extensible and the rest of the constructs as inextensible. This means that if a new exam item type is added as a specialization of *ExamItem*, the content model of the corresponding target XML schema component remains unchanged and can still be reused by the specialization. Class *Exam* is considered extensible because the addition of a new exam item type must not affect the content model of the schema component to which *Exam* is mapped.

Based on this decision a selection operation is used to reduce the space $S_{ExtensibleExam}$:

$$S_{ReducedExtensibleExam} = \text{Select from } S_{ExtensibleExam} \text{ where } \langle \text{ExamItem.Exts and Exam.Exts and Open.InExts and MultipleChoice.InExts and ExamItem\_Open.InExts and ExamItem\_MultiChoice.InExts and Exam\_ExamItem.InExts} \rangle \quad (13)$$

The space $S_{ReducedExtensibleExam}$ identifies extensible constructs from the source model. Now we aim at target models that guarantee the extensibility property of the resulting constructs. Extensibility properties are integrated into the transformation space $S_{ReducedExamSchema2}$ using operation merge defined over two transformation spaces.

Operation merge has two transformation spaces S_1 and S_2 as arguments and returns a new space:

$$S_{merged} = merge(S_1, S_2) \quad (14)$$

where $dimensions(S_{merged}) = dimensions(S_1) = dimensions(S_2)$. The coordinate set of a dimension d in S_{merged} is a Cartesian product defined in the following way:

$$coordinateSet(d, S_{merged}) = coordinateSet(d, S_1) \times coordinateSet(d, S_2)$$

The points from S_{merged} are defined on the basis of the points from S_1 and S_2:

S_{merged}={(d_1.(p_1', p_1''),…,d_n.(p_n', p_n'')) | $d_i \in$ dimensions(S_{merged}),
p_i' $\in$ coordinateSet(d_i, S_1), p_i'' $\in$ coordinateSet(d_i, S_2),
($d_1.p_1$',…, $d_n.p_n$') $\in$ S_1,($d_1.p_1$'',…, $d_n.p_n$'') $\in$ S_2 for i=1,…,n}

Operation merge may be applied on the spaces $S_{ReducedExamSchema2}$ and $S_{ReducedExtensibleExam}$ to create a transformation space for extensible exam schemas $S_{ExtensibleExamSchema}$:

$S_{ExtensibleExamSchema}$ = merge($S_{ReducedExamSchema2}$, $S_{ReducedExtensibleExam}$) (15)

Now the software engineer has the quality properties explicitly represented and can use selection criteria based on them. In the process of space reduction various knowledge sources may be used, for instance, heuristic rules. In our example, we identify the problem of extensible schemas as the problem of containers with variable contents described in the XML Schema Best Practices [Cos03]. The solution proposed there may be summarized in the following heuristic rule:

```
IF extensibility is required for a container class that aggregates other
classes, which are specializations of a common general class, THEN
  define an element (E) for the container class,
  define an element (E) for  each specialized class,
  define an abstract element (E_abstract=true) for the general class and
  define substitution (Subst) between the general class element and each
specialized element
  OR
  define an element (E) for the container class,
  define an element (E) for the general class,
  define complex type (CT) for each specialized class,
  define an extension (Ext) between the type of the general class and the
type of each specialized class.
```

From this rule the software engineer may construct a condition used in a selection operation over $S_{ExtensibleExamSchema}$. The resulting space contains two alternatives shown in Table 2:

N	Exam	ExamItem	Open	MultipleChoice	ExamItem_Open	ExamItem_MultiChoice	Exam_ExamItem
1	E	E	E	E	Subst	Subst	Cont
2	E	E	CT	CT	Der	Der	Cont

Table 2: Two Alternatives for Extensible Exam Schemas.

It can be noticed that these alternatives are similar to alternatives (c) and (b) shown in Figure 4.

3.4 Refinement

This is the last step of the process for the construction and reduction of a transformation space. Once the space has sufficiently been 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 can be specified and executed. For instance, the element declaration components always require a type to be defined. Substitution relations between element declarations imply derivation between the corresponding element types. In addition, in the example presented here it was decided not to include the attributes of the classes in the transformation space. Certain decisions must be taken as to how the attributes are to be mapped.

4 Discussion

We have presented techniques for the construction and reduction of transformation spaces for a given source model. In this section we evaluate the approach with respect to the complexity of the transformation spaces and the applicability of this technique in the context of model transformations as defined in MDA.

Complexity of the transformation space. Although transformation spaces tend to be rather large even for simple models, they are purely conceptual. The software engineer does not need to generate the alternative transformations from a transformation space unless a number of reduction steps are applied and the size of the space is reduced sufficiently. 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 individual alternative.

There is an analogy between the concepts of transformation space and relational schema used in relational databases. In the case of a relational schema, for example, the user does not have to deal with the concrete data set of the corresponding database, which is usually dynamic. Instead, the user specifies queries based on the structure of the relational schema. Similarly, the concept of a transformation space provides the means to specify the conditions to reduce and select alternatives from a transformation space.

Apart from selection and exclusion, other techniques may be applied to reduce the complexity of a space. After the necessary reduction steps, the model may be decomposed into parts that are isolated from each other. In this case, alternatives from isolated parts do not influence each other. In the example presented here, we can assume that the content model of Alternative and Question is somehow encapsulated within the schema component. Then, the part of the source model comprised of classes Question, Alternative and Content and the relations among them may be treated separately from the rest of the model. We are currently investigating the applicability of this idea and the required operational support for this.

Applicability of the technique in the context of MDA. Two types of models are specified in the context of MDA: PIMs and PSMs. Four categories of transformations are possible depending on the source and target models: PIM to PIM, PIM to PSM, PSM to PSM and PSM to PIM [MM01]. In general, the concept of transformation space may be applied in every one of these categories. The example described in section 3 presents a transformation from a PIM expressed in an UML class diagram to a PSM based on an XML Schema.

In this paper, the coordinate sets are based on a one-to-one mapping from the constructs of the meta-model of a source to the constructs of the meta-model of the corresponding target. This approach, however, has to be extended if the constructs of a source model are mapped to several constructs of the corresponding target.

5 Related Work

The work presented here is an adaptation of a formalism called Design Algebra [AT02] used for the identification of design alternatives for a given design problem. From the perspective of model transformations Design Algebra supports the construction of a transformation space for models specified as platform independent software solutions. The target meta-model contains the constructs found in the traditional object-oriented languages. In this paper this technique is generalized to support transformations between arbitrary models in the context of MDA where the source and target models conform to given source and target meta-models that provide the information for construction of transformation spaces.

The problem of deriving XML schemas from UML class models described in the example is addressed in [BGH00, Car01, EM01]. The authors identify the presence of multiple target schemas that differ in their quality characteristics. In [BGH00], generated schemas must ensure minimum data redundancy and maximum connectivity in the documents. This is achieved by a proper construction of the document hierarchy. Models are transformed by an algorithm based on 12 heuristic rules. The method described in [EM01] also aims at producing schemas that ensure minimum data redundancy in the documents. The authors give a formal definition for the minimum data redundancy property named canonical normal form of XML documents (XNF).

In these papers the problem of identifying the alternative transformations is not addressed. Instead, algorithms are defined that produce results with certain quality properties. We consider

these contributions as complementary to our work. The knowledge they provide can be incorporated and applied during the reduction process.

Our technique is also complementary to existing transformation languages and can precede the specification and execution of transformations.

6 Conclusions

Transformation between models is a key operation in the OMG's MDA vision for software development. Generally one may adopt different but functionally equivalent target models for the same source model. Functionally equivalent target models, however, may differ from each other in the quality properties they possess. For example, one target model may be more extensible than the other target models. 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.

In this paper we proposed a method for identifying a set of target models (the transformation space) for a given source model. This method requires a source model, its meta-model and the meta-model of the target as input and generates a transformation space defined by the alternative target models as output. Reduction of transformation spaces is supported by selection and exclusion. These operations can be parameterized by the quality attributes and be supported by heuristic rules. In this way, the software engineer is able to select the desired target model from the transformation space. Currently, we are carrying out several case studies to evaluate the applicability of the method in different categories of model transformations.

References

[AT02] M. Aksit, B. Tekinerdogan: *Deriving Design Alternatives Based on Quality Factors.* Kluwer Academic Publishers, In Aksit, M. (ed.) 'Software Architectures and Component Technologies'; 2002.

[BGH00] L. Bird, A. Goodchild, T. Halpin: *Object Role Modeling and XML-Schema*. In Int. Conf. On Conceptual Modeling (ER), Salt Lake City, UT; 2000.

[Car01] D. Carlson: *Modeling XML Vocabularies with UML*. Addison-Wesley; 2001.

[Cos03] R. Costello: *XML Schemas: Best Practices*. Available at: http://www.xfront.com/BestPracticesHomepage.html

[DMO01] S. DeRose, E. Maler, D. Orchard: *XML Linking Language (XLink).* W3C Recommendation; 2001.

[EM01] D. Embley, W. Y. Mok: *Developing XML Documents with Guaranteed "Good" Properties.* 20th Int. Conf. on Conceptual Modeling (ER), Yokohama, Japan; 2001.

[MM01] J. Miller, J. Mukerji: *Model Driven Architecture (MDA)*. OMG Document available at http://www.omg.org; 2001.

[OMG00] OMG/MOF: *Meta Object Facility (MOF) Specification*. OMG Document. 2000.

[OMG01a] OMG/UML: *OMG Unified Modeling Language Specification*. OMG Document. 2001.

[OMG01b] OMG/XMI: *XML Metadata Interchange (XMI) Version 2*. OMG Document. 2001

[TBM01] H. Thompson, D. Beech, M. Maloney, N. Mendelsohn: *XML Schema Part 1: Structures*, W3C Recommendation. http://www.w3.org/TR/xmlschema-1; 2001.