Some Model Transformation Approaches: a Qualitative Critical Review

May Dehayni, Kablan Barbar, Ali Awada, Mohamad Smaili

Department of Applied Mathematics, Lebanese University, Faculty of Sciences I, Hadath, Lebanon

Abstract: Model-to-model transformation is a critical activity in Model Driven Engineering. In this paper, we will classify the existing model transformation approaches and propose a set of criteria characterizing different approaches. We will discuss the criteria according to their transformation expressivity power and implementation and parsing strategy techniques. We have proposed in [1] a system for the transformation of information system models based on the formalism of attribute grammars. This formalism has been developed by Knuth [2,3] for the specification and implementation of static semantic aspects of programming languages. Since then, it has matured into a recognized field of study with many applications [4,5]. The goal of this contribution is to guide the model transformer when it comes to undertaking a transformation, while taking into account some existing facilities and constraints.

Key words: meta-modeling; MDA; model transformation; XML; XSLT; Attribute grammars; QVT.

INTRODUCTION

The OMG (Object Management Group) has proposed a four layer reflective framework based on the MOF (Meta Object Facility) [6] architecture in order to allow the definition of meta-models. The MDA (Model Driven Architecture) aims to separate application structure PIM (Platform Independent Model) from its functionality, PSM (Platform specific Model). The mapping between these models is realized by model transformation. The problem of model transformation based on the MOF can, then, be stated in the following way: “Given a source model ‘m1’ described by a meta-model ‘MM1’, we define an automatic process making it possible to obtain a model ‘m2’ conforming to a meta-model ‘MM2’; ‘MM1’ and ‘MM2’ being MOF compliant.” (see Figure 1).

In fact, this process is similar to the one applied in compiler construction, in which a source text, obeying some syntax and semantic rules, is transformed into a target text conforming to another set of syntax and semantics. Model-to-model transformations are of primary importance in the evolution of information system meta-models, for the purpose of keeping the different models of a system up-to-date, facilitating cooperation among developers, amalgamating models, and developing new approaches such as Transformation-based Model Generation (TMG) which enhances the productivity of application development [7], handling conversion between XML documents and relational data used to store critical business data [8]. The need for standardization of a transformation language led to the MOF 2.0 Query/Views/Transformations (QVT) [9] standard, which is one possible approach to solving model transformation problems. However, it is still unclear which model transformation approaches are best suited for which applications, and, further, whether it will be possible to express all possible model transformations with a single approach. It seems likely that a number of different approaches will be needed in the future. As a consequence, it is important to know the advantages and disadvantages of these different approaches, to make it easier to choose the right approach for a given application [10]. This paper consists of a presentation of the following transformation approaches: graph transformation [11], transformation using the Application Programming Interface (API), the XSLT approach, the declarative approach, the hybrid (or dedicated) approach, and the AG approach. We then present a brief summary of transformation approach [12,1] and conclude with a study comparing these approaches, according to a set of criteria based on the expressivity power and implementation and parsing strategy criteria characterizing model transformation approaches.

Some Approaches to Model Transformations: In this paper, we will classify the existing model-to-model transformation approaches into graph transformation based approaches, direct manipulation approaches via API, XSLT based approaches, relational or declarative approaches, hybrid approaches and attribute grammar based approaches.
Graph Transformation Based Approaches: Generally, (meta-)models are represented in UML formalism. As a result, the models can be viewed as graphs. It is therefore natural to consider the use of graph grammars \[13\] to express model transformation. Graph transformation approaches are very well founded theoretically. They favor matching and replacement \[14\]. They are based on syntactic graph rules that consist of finding a Left Hand Side graph and replacing it by a Right Hand Side graph \[15\]. This approach has the power of a clear operational idea, which enhances rule specification \[16\]. The complexity of this approach stems from its non-determinism in scheduling and application strategy, which requires careful consideration of termination of the transformation process and the sequence of rule application. Thus, in the remainder of this paper, we will focus on transformation methods using a textual model representation. This format is the one used by the standardized XMI format.

Direct Manipulation via API: Meta Object Facility specification is used in many modelling tools to create model repositories and generate Application Programming Interfaces for each supported meta-model \[17,18]\]. These interfaces are used to describe the model transformation process by means of programs written in an imperative language: Java, C++, Eiffel, Python, etc. This approach provides the user with a set of interfaces used to describe the transformation process as a series of instructions that allow the generation of a target model from a corresponding source model. The use of APIs to describe a transformation process is a powerful solution because programming languages generally have good performance at runtime. Basically, the entire procedure must be performed by the user: he is in charge of the organization and description of all stages, explicitly in terms of imperative statements.

Xslt Based Approaches: The XMI standard offers a format for the exchange of meta-models. This format is generated by a DTD (Document Type Definition) in XML, which defines the representation of all meta-models. Relations between a DTD and an XML document are similar to those between a meta-model and the instantiated models. Both of them involve a word (XML document / model) of a language, generated by a grammar (DTD / meta-model). As models are XML texts, it appears that XSLT is a convenient solution for model transformation. XSLT is an appropriate standard for XML document transformation, but suffers from limitations \[17,18]\ in realizing model transformation. Moreover, XSLT data types are limited; this restricts the scope of information that must be computed during the transformation process. In a DTD, the syntax and the semantics of an XML document are mixed, and transformation rules must therefore deal with both.

Declarative Approaches: The relationship between concepts in the source and the target meta-model is defined by patterns. The transformation is defined by a set of rules. A rule lays forth a pattern of source model concepts, which is then transformed into a set of elements in the target model. The sequence of the various stages of the transformation process is controlled by the user, thanks to operators that allow the carrying out of explicit transformation rules invocation. The implementation is realized by an inference engine, as in MIA, (Model In Action) \[19\] based on Prolog.

Hybrid (Or Dedicated) Approaches: In a declarative approach, a transformation is defined by a set of relations between the concepts of the source and target models, as described in their meta-models. The implementation is realized by an inference engine, which allows the application of the transformation to generate the target model. In an imperative approach, a transformation is described by a set of algorithms as functions or procedures that explicitly describe the sequence of transformation applications. Hybrid approaches combine the declarative and imperative approaches; as in \[18,19,20\]. The declarative approach is generally used in the definition and selection of the transformations which can be applied, while the imperative approach is well adapted to describing the transformation strategy by a control flow of execution rules, and hence to executing the transformation. In hybrid languages, transformation rules mix the syntax and the semantics of the concepts they handle.

Model Transformation Based on Attribute Grammars: In our proposal, we have chosen to avoid developing a new transformation language, and rather to explore a way that is formally founded. A meta-model may be viewed as a grammar which generates a language that describes models. This process is similar to the one applied in compiler construction, in which a source language, obeying some syntax and semantic rules, is transformed into a target language conforming to another set of syntax and semantic rules. Our approach to model transformation was inspired largely by this similarity. Therefore, compilation theories and techniques were essential to solve the problem. We have proposed a transformation strategy based on the AG (attribute grammars), which is both theoretically founded and operationally equipped.

An AG consists of a context-free grammar extended by a set of attributes and rules for their computation, which specifies the model’s semantics. It
is based on an abstract syntax (AS) of meta-models. Our transformation approach is essentially based on attribute declarations and their semantics rules, allowing the computation of the attributes on a syntactic tree; the result here is the generation of the target model. The semantics in this proposal consists of realizing model transformation through an ascending generation of the target model: the target model is synthesized at the root of the syntactic tree.

The implementation is realized by a system called an "evaluator." In our study we have chosen to use the "Cornell Synthesizer Generator" (CSG) \(^{[1]}\). This system is based on an incremental attribute evaluator that can process AGs belonging to the class of "Ordered Attribute Grammars" (OAG) \(^{[2,3]}\). This choice appears to be a good compromise between the expressive power of processed AG and runtime performance.

The first step consists in defining the AS of a textual format representation of the (meta-) models and providing an AG to describe the transformation. The transformation framework generates a textual representation of the target model. Its input is the text that describes the source model, the syntax of which is defined by an AS of the source meta-model. The output is another text, corresponding to the specification of the target model in accordance with its grammar.

AG formalism makes it possible to locate information essential for the transformation of source concepts by the means of attributes attached to the corresponding non-terminals. At this stage, we may define the "context" of a concept "C" as the attributed sub-tree of which "C" is the root. Thus, a concept "C" of a source model can have: (Figure 2)

1. Synthesized attributes for the information that is calculated from its context, i.e., the values of its properties and the contexts of its descendants;
2. Inherited attributes for the information calculated out of the context of "C", i.e., from the contexts of its siblings, its parents and the other concepts of the source model which are not derived in the syntactic tree.

Model transformation based on AG is directed by the elements of the source model. Thus we call the concept that directs the generation of target element(s) the "directing element" (abbreviated "D.E").

Among the most significant classes, we can distinguish: the purely synthesized grammars (in which the attributes can be evaluated during bottom up syntactic analysis), the Left Attribute Grammars (LAG) (in which it is possible to evaluate the attributes during the syntactic analysis), the OAG (in which the sequence of attribute computation is total and defined by the grammar), and the Strongly Non-Circular (SNC) AGs (in which the order is fixed dynamically on each syntactic tree) \(^{[4]}\). There exist several AG evaluators. Each of them has a language that expresses its semantic rules. Although SNC are the most expressive, they are also less effective at evaluation than those other classes which are more constrained. Consequently, it is essential to choose for the transformation an expressive AG which is also the most effective in evaluation.

Model transformation based on AG is directed by source model elements. Therefore, we have defined the "directing element" of a transformation of one or more concepts of a source model as any concept that directs the generation of one or more corresponding target elements.

In the following discussion, we will describe the choice of the "D.E" of the transformation through an analysis of different source and target element correspondences. Once the "D.E" is selected, we must associate it with a set of attributes that calculate the information needed for the transformation. In the case of:

- A simple correspondence (1 to 1), and in decomposition (1 to N): the source element is a "D.E."
- Multiple correspondences (M to N): we propose to initially amalgamate (element amalgamation is described later) the M source elements into one element, which is the "D.E," and then to decompose it into N target elements.
- Element amalgamation: we have to face several choices for the "D.E." We propose to choose this element whose passing cost is the least expensive. Let us consider an example of the fusion of three concepts, "A," "B" and "C," of a source model where A and B are siblings and C is a child of A. These elements will generate a target concept "D."

We will analyze the potential choices of the "D.E" from the point of view of the impact on the AG class. We have three possibilities:

- With "A" as the "D.E," the information of "C" will be passed using synthesized attributes and that of "B" using inherited attributes; the grammar is OAG.
- If "B" is the "D.E," the information of "A" and "C" will be passed by inherited attributes; the grammar is LAG.
- If "C" is the "D.E," the information of "A" and "B" will be passed by inherited attributes; the grammar is OAG.

The LAG class has an evaluation algorithm more efficient than that of the OAG class. Thus, we will choose the second case for this fusion example.
Fig. 1: Model transformation based on AG

Fig. 2: Attributes declaration of a source concept

<table>
<thead>
<tr>
<th>Table 1: Transformation expressivity power criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Criterion</strong></td>
</tr>
<tr>
<td><strong>Approaches</strong></td>
</tr>
<tr>
<td>XSLT</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Hybrid or dedicated language</td>
</tr>
<tr>
<td>MTrans</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>TRL</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>ATL</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Attribute Grammars</td>
</tr>
</tbody>
</table>
Transformation Expressivity Power Criteria: In this section, we will present a non-exhaustive list of criteria characterizing model transformation approaches. Some of these properties fit the requirements of the QVT [9]. Although this list is non-exhaustive, it has appeared significant in existing or desirable properties for the transformers; it enriches the criteria proposed by the example in [24]. These criteria will enable us to discuss the contributions of our approach by a qualitative comparison with other approaches. (Table 1)

Reusability: Let “T1” be an existing transformation. We wish to obtain another transformation, “T2,” through an adaptation of “T1”, i.e., without completely redefining “T2.” We may reuse the property which makes it possible to obtain “T2” starting from “T1” by adapting the process establishing “T1.”

For a model transformation approach we must distinguish the rules reused within a transformation--which we call "reuse by rule adaptation"--from the reuse of the totality of a transformation to build others--which we call "reuse by transformation composition."

Rule Adaptation: Reuse mechanisms of programming languages are implemented by composition, inheritance or modularity via functions or procedures. These mechanisms are also used in model transformation approaches: Composition is used in MIA [23] and VMT [26], and is recommended in [27]. It is planned in ATL (Atlas Transformation Language) [18] and TRL (Transformation Rule Language) [20]. Inheritance is implemented in the hybrid languages MTrans (Model Transformation) [19], TRL and ATL. Parameterized rules are envisaged in ATL and TRL. Transformation functions are used in our approach [12,1] by the CSG, which uses the procedural language SSL [21] close to C language, which makes it possible to describe transformation functions and to re-use them by calls.

Transformation Composition: Transformation composition makes it possible to build complex transformations from elementary ones. In MIA [25], the composition of two transformations, “T1” and “T2,” can be done by using two sets of rules, “R1” and “R2,” respectively. Taking into account the declarative character of MIA, the new goal can be obtained by carrying out a fusion “R3” from “R1” and “R2.” It is advisable, however, to respect a sequence in “R3” which ensures that “T1” is executed well before “T2.” In hybrid languages a rule is a transformation step. To obtain “T3,” it is necessary to connect the rules which establish “T1” with those of “T2.” The first rule of “T2” must succeed the last rule of “T1.” The ATL developers [18] are currently working on the definition of HOT (Higher Order Transformation), which is a way to generate new transformers using existing ones. In our approach, transformation composition is theoretically ensured by the formalisms of HOA (Higher Order Attribute Grammar) [24] and ACG (Attribute Coupled Grammar) [29,30]. These theoretical results are not currently supported by all AG evaluators. They require that the evaluators be equipped with particular appraisers integrating this possibility, which is not the case with the CSG used in our work.

Traceability: Traceability is the property which makes it possible to store all the information considered relevant concerning correspondences between source and target elements, to record links between their source and target elements as well as the various stages of the transformation process. Traceability links can be stored in the target model or separately. Some approaches provide dedicated support for traceability which can create traceability links automatically, as in hybrid languages MTrans [19,20] and ATL [18] which possess a traceability model. For each execution, the transformation engine generates a model in conformity to the traceability meta-model. In these languages, a transformation rule is a transformation step. This allows, in particular, the development of new executions, the cancellation of some stages, etc. Our approach allows the user to encode traceability. Indeed, all desired computable information can be obtained by declaring adequate attributes and calculation functions. On the other hand, the concept of a “transformation step” such as that defined in hybrid languages does not exist; thus it is not possible to replay or cancel a step.

In-place Transformation: An in-place transformation or model update is a transformation in which the
source and target meta-models are the same. In all the existing approaches, the update consists of handling two meta-models—which are in this case identical—as well as the source model, in order to generate the target model, which is always a new one.

In ATL \cite{31} the source model is first copied to the target model, and then transformation rules are applied. Such a transformation model can be thought of as having a set of implicit rules, which copy all source model elements, and a set of explicit rules written by the modeller. In our approach, the AG handles only one tree structure of the source model. Thus the update is done on this tree, and the in-place transformation is then supported by an incremental mechanism, which we will describe below, which is established in some AG evaluators such as the CSG that we use.

**Incremental Transformation:** A transformation is considered incremental if any change in the source model causes only the execution of those transformation rules associated with the modified concepts. The hybrid languages offer no direct support for incremental transformations. However, they could be implemented using traceability. This remains an area of ongoing research \cite{28}. An AG offers incrementality by evaluation \cite{4,21}. The CSG generates an incremental evaluator. This property offers the possibility of replacing an attributed sub-tree with another; the new attribute values would be automatically propagated on the entire syntactic tree.

**Reversibility:** A transformation “T1” is reversible if there is a transformation “T2” such that the application of “T1” followed by “T2” is the identity function. This property is very useful in cancelling the effects of a transformation.

Currently, none of the transformation approaches which we have studied make it possible to automatically deduce the totality of the reverse transformation. However, ATL as described in \cite{31} makes it possible to automatically reverse some rules of a transformation using the HOT process.

**Conclusion:** The success of model transformation technology will depend on the existence of multiple languages for specific model transformations used in various domains, in which detailed characteristics are needed to help the user to decide which approach to use in a certain application. With the current amount of model transformation approaches, the question that arises is which is best suited for which application, and where the differences are \cite{6}. In Table 1 we summarize the above discussion about transformation expressivity power criteria and draw a comparison between the different model transformation approaches.

As noted in the comparison table, we have concluded that direct manipulation via XSLT is obviously the most low-level approach. All the transformation work must be completed by the user.

Hybrid approaches allow the user to mix and match different concepts and paradigms depending on the application. They propose dedicated model transformation languages through a set of instructions. Model navigation is generally described in OCL (Object Constraint Language). Such an approach proposes a new syntax and new concepts; therefore it is necessary to spend some time learning the language. Nevertheless, the hybrid languages propose a set of shorter instructions, and are generally more declarative than imperative; therefore, training time is reduced. These languages can be interpreted or compiled; indeed, if the implementation is realized by a compiler, then the performances are not bound directly to the transformation language but rather to the compiler's target language \cite{30}. MTrans, TRL and ATL are compiled to widely used programming languages (Java, Python, etc.). These languages allow the direct description of the transformation algorithm, which are difficult to express in transformation language. If the target language is interpreted, then the performance is less effective than that of compiled target language.

The AG approach is a purely declarative; we declare semantic rules that achieve the model transformation without taking into account their application order. This order will be defined by an evaluator that compiles the AG specification in an executable model transformer. The use of AGs for model transformation offers the traditional benefits of this formalism—being theoretically based, an AG guarantees precision. Semantic rules are expressed in a declarative mode on the syntactic structure, while transformations are expressed concisely and clearly without being overburdened with code. The modularity of a specification, written in AG, is ensured by its block structure, derived from grammar productions. An AG is structured by blocks that contain grammar productions. It induces a modularity definition for every source model concept as well as the principle of encapsulation inside the blocks. This allows the specification of local production computations: the attributes access is strictly local to the production attributes' occurrences, and the inherited and synthesized attributes allow dataflow between attributes of different productions. Each source concept that must be transformed has a synthesized attribute that generates its target concept. Therefore, pattern matching doesn't exist as it does in most other approaches, like XSLT, MTrans, TRL, and ATL.
Transformation Implementation and Parsing Strategy Criteria: XSLT is an appropriate standard for XML document transformation, but it is not very convenient for model transformation for the reasons quoted in [17,32]. Moreover, XSLT transformation is based on top-down or bottom-up parsing on XML syntactic tree of the source model. Furthermore, XSLT has limited calculation information on a tree. Thus, a realization of several traversals on the source model tree--essential to search the information needed to accomplish the source entity transformation--necessitates the creation of intermediate trees of intermediate computations, which can make the transformation process less powerful, difficult to read, and very redundant.

In hybrid languages, transformation rules mix syntax and semantics. Source model parsing and rules invocation order are carried out by the user. The model transformation approach, based on AG, allows the gathering of the information required to accomplish the transformation of the source entities. This information is assembled by attributes associated with the nodes representing source model entities on the derivation tree. This enables the user to avoid making many in-depth traversals of the source model tree structure in order to seek necessary information. An AG allows explicit separation between source model syntax and semantics.

Direct manipulation doesn't offer the user any support or guidance in implementing transformations. The use of XSLT requires the user to explicitly program the necessary tree traversals in order to obtain a target model. Moreover, XSLT is processed by an interpreter. The programmer of a model transformation in a hybrid language must explicitly control the various stages of the process, by invoking suitable rules and by implementing functions that permit to the user to make deep traversals of the source model structure in order to seek required information [12]. The AG approach separates model syntax and semantics. It is based on the AS model, which implies a less cumbersome memory representation and a more efficient implementation compared to other approaches based on the concrete syntax model. AG formalism constitutes an executable method of specification, as it simply describes a computation in terms of an AG and then automatically produces a program which carries it out. Since the evaluator code is automatically obtained, it follows that effective implementation is very simple. The implementation of our model transformation approach is done in the Cornell Synthesizer Generator [21], which requires the language (i.e. meta-model) syntax specification and the computation rules to generate an automatic evaluator for the semantics that is the model transformation in our survey. It generates incremental compiled evaluators. This property offers the possibility to replace a sub-tree of a syntactic tree representing a model with another sub-tree. The propagation of new attribute values on the whole tree is then processed automatically. Thus, using such an incremental evaluator allows the introduction of reusability into the model transformation. In certain cases, the definition of a new model transformer can be obtained by a mere attributed sub-trees substitution [12].

To summarize the above discussion about Transformation Implementation and Parsing Strategy Criteria, a comparison among the different approaches is shown in Table 2.

Conclusion: In this paper we have attempted to classify the existing model transformation approaches. Our work has mainly consisted of transposing the model transformation problem into language compilation. This has led us to choose AG to propose our model transformation architecture. The use of AG to carry out model transformation is a purely declarative approach which automatically generates the transformer as an AG evaluator. This approach is based on meta-model abstract syntax, which ensures facility of handling and greater effectiveness in implementation than concrete syntax.

One of the principal advantages of our approach is the possibility of classifying the transformations using the AG. This classification is a measure of the complexity of the transformation. It offers the user the possibility to rewrite the AG describing the transformation and put it in a more efficient class, and to thereby obtain a more powerful transformer.

However, model transformation based on AG is an important task. It consists of the definition of the lexical analysis, abstract and concrete syntax of the meta-models, as well as attributes and their evaluation rules. These different parts are described in separated files. The problem related to the code size could be solved on evaluators. For this reason, we think in particular of the following perspectives for the continuation and improvement of our work: In our work, we have used the CSG evaluator for model transformer construction. The advantages of this software are numerous, both theoretically and operationally. However, the major disadvantage of the CSG, and generally of the current version of evaluators, is the lack of interactivity with the user. It is the user’s responsibility to explicitly describe all description stages of an AG. It appears essential to equip evaluation systems with an interactive mechanism, which would allows users who do not possess thorough knowledge of AG to use our system and to be guided through the realization of the model transformers.
Meta-models are currently described in XML or other proprietary languages. Our approach is based on AS formalism of meta-models. The AS of a DTD/XML of a meta-model is manually laid out in our transformer. These rules must be implemented using a tool that can generate the AS of a meta-model, relying, for example, on algorithms depicted in [31].

REFERENCES


6. www.dstc.com/Products/CORBA/MOF


9. MOF 2.0 Query/Views/Transformations RFP. OMG document, ad/03-03-03.


11. Zündorf, A., 1996. Graph pattern matching in PROGRES. In the proceedings of the 5th International Workshop on Graph-grammars and their application in computer science, LNCS 1073, USA.


