Proceedings of the 2014 Federated Conference on Computer Science and Information Systems pp. 1601–1610
DOI: 10.15439/2014F144
ACSIS, Vol. 2

Grammar-Based Model Transformations

Galina Besova
Department of Computer Science
University of Paderborn
33098, Paderborn
Email: besova@mail.upb.de

Dominik Steenken
Department of Computer Science
University of Paderborn
33098, Paderborn
Email: dominik@mail.upb.de

Heike Wehrheim
Department of Computer Science
University of Paderborn
33098, Paderborn
Email: wehrheim@mail.upb.de

***Abstract*—Model transformation is a key concept in model-driven software engineering. The definition of model transformations is usually based on meta-models describing the abstract syntax of languages. While meta-models are thereby able to abstract from superfluous details of concrete syntax, they often loose *structural* information inherent in languages, like information on model elements always occurring together in particular shapes. As a consequence, model transformations cannot naturally re-use language structures, thus leading to unnecessary complexity in their development as well as analysis.**

In this paper, we propose a new approach to model transformation development which allows to simplify and improve the quality of the developed transformations via the exploitation of the languages' structures. The approach is based on context-free *grammars* and transformations defined by *pairing* productions of source and target grammars. We show that such transformations exhibit three important characteristics: they are *sound*, *complete* and *deterministic*.

I. Introduction

MODEL transformations are key to model driven engineering (MDE). Surveys on model transformations [1], [2] show their expanding application areas: model translation, model composition, refinement, and other.

In an MDE setting, the syntax of models is given in terms of *meta-models* which themselves conform to their own meta-models (e.g., MOF [3]). Meta-models define the *abstract syntax* of languages, abstracting away from the details of concrete syntax like keywords and ordering of elements. Model transformations thus operate on abstract syntax. While meta-models describe model elements and their direct relations, they fall short of describing more complex interrelations like sets of model elements always occurring together in particular shapes. In some cases, meta-models are enriched with OCL [4] constraints to enforce such shapes in models.

In contrast to MDE, traditional approaches to language definition (and translation) define languages by *grammars*, often given in an Extended Backus-Naur Form (EBNF) [5]. These translation techniques operate on concrete syntax. While the details of concrete syntax are in general unimportant (and thus make translation definition unnecessarily confusing), the *structural* information contained in the grammars is highly useful for defining translations. The productions of the grammars define the structures available in the languages, and by relating productions of grammars (as done in syntax-directed translation [6]) we can easily specify how language structures are mapped onto each other.

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901).

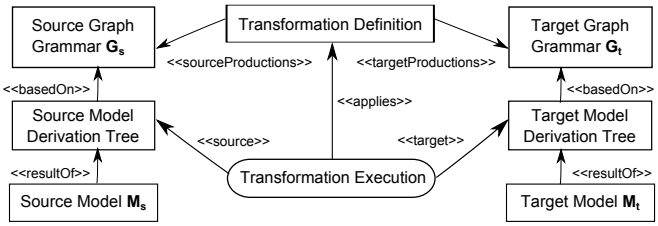


Fig. 1: Overview of our approach

An ideal approach for model transformation should thus combine these two approaches, taking the best of both: have language definitions with the abstract syntax of meta-models and the structures of grammars, and build model transformations on these definitions. An early approach following this idea, although not in the area of model transformations and not with meta-models but with graphs, is the one of Pratt [7]. Pratt defines *pair grammars* as a way of relating the grammars of two languages, thus obtaining a natural way of relating languages and building translations from one to the other.

The objective of this paper is to bring the idea of pair grammar based translation to the world of MDE and model transformations, lifting it to the level of abstract syntax while preserving its advantages. We also extend it in order to cover a broader variety of model transformations.

Fig. 1 gives an overview of our approach. The transformations we focus on are model-to-model transformations. Our models are given in abstract syntax and are generated by grammars. For this generation purpose we use a type of context-free graph grammars – *hyperedge replacement graph grammars* [8] – typed and constrained by meta-models. Transformation rules – like pair grammars – relate productions of the source with those of the target grammar. Model transformations are executed on the *derivation trees*: given a source model $\mathtt{M_s}$, its derivation tree in the source grammar is obtained by parsing, and used by the model transformation to produce a derivation tree in the target grammar, and thereby the corresponding target model $\mathtt{M_t}$.

We exemplify our approach on a transformation from activity diagrams to the process algebra CSP [9]. We also prove

important qualities of the transformations developed with our approach – *termination*, *soundness*, *completeness*, and *determinism*. Showing these quality properties for a transformation described using current state-of-the-art techniques is usually hard, if not impossible [10].

First, in Sec. II we give background on grammar-based language definition and show our source and target grammars. Then, we introduce our approach in Sec. III and in Sec. IV show the quality characteristics of the developed transformations. In Sec. V we demonstrate extensions of our approach, and in Sec. VI we evaluate it in comparison with the most closely related approaches in MDE. Finally, we survey related work in Sec. VII and conclude in Sec. VIII.

II. Background

There are two fundamentally different ways of specifying the syntax of a given language: with (context-free) grammars or with meta-models. Our approach is built on grammars generating instances of meta-models, i.e., graphs. In the following, we introduce the main concepts of the grammar-based language definition and show how they can be lifted to graph-based languages, enabling grammar-based definition of modelling languages and model transformations utilizing these definitions. We show how our example modelling languages for activity diagram and CSP can be described using such grammars. Finally, we introduce the transformation example used later to demonstrate our approach.

A. Grammar-Based Syntax Definition

In their original usage, grammars define languages of strings via a set of generative rules. We briefly review some general definitions of grammars for string languages [6] because our graph grammars is a natural extension of string grammars.

Definition 1 (Grammar)**.** *A* grammar $G = (N, \Sigma, P, S)$ *consists of a set of* non-terminal symbols N*, a set of* terminal symbols Σ*, a set of* productions P *and a designated* start symbol $S \in N$*. Each production* $p \in P$ *is of the form* $p = (l, r)$*, with* $l \in (\Sigma \cup N)^* N (\Sigma \cup N)^*$*, i.e., a string of symbols with at least one non-terminal, and* $r \in (\Sigma \cup N)^*$*.*

A grammar $G = (N, \Sigma, P, S)$ *is called* context-free *iff every production* $p \in P$ *has the form* $p = (n, r)$ *with* $n \in N$*.*

Applications of productions derive new strings from given ones by a process called *rewriting*. A string $s \in (N \cup \Sigma)^*$ is rewritten into a new string by a context-free production $(n, r) \in P$ by finding n in s and replacing it with r. In this way, a grammar defines the set of strings that can be *derived* from its start symbol S. This set is called the *language* of the grammar G.

Definition 2 (Language)**.** *Let* $G = (N, \Sigma, P, S)$ *be a grammar. A* sentence *of* G *is a string* $s \in \Sigma^*$ *of* terminal *symbols that can be derived from* S *using a finite sequence of applications of productions in* P*. The language* $\mathcal{L}(G)$ *of* G *is defined as* $\mathcal{L}(G) = \{s \in \Sigma^* | s \text{ is sentence of } G\}$

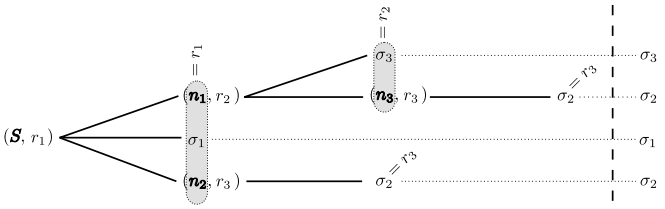


Fig. 2: A derivation tree deriving $\sigma_2\sigma_1\sigma_2\sigma_3$ from S

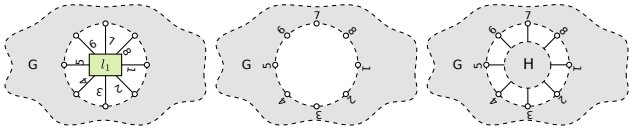
The advantage of context-free grammars is that parsing can be done efficiently [6]. Parsing a sentence in a language using a context-free grammar gives us a *derivation tree*, i.e., a structure showing the application of productions from the start symbol leading to the derived sentence. A derivation tree for a sentence defines its *structure* and confirms its language membership in $L(G)$ (i.e., syntactical correctness). The inner nodes of such a tree are labelled with productions. The root is labelled with a production that consumes the start symbol of the grammar. Every leaf of the tree is labelled with a terminal symbol. Fig. 2 shows an example derivation tree for the sentence $\sigma_2\sigma_1\sigma_2\sigma_3$.

In general, there can be multiple derivation trees for a sentence in one grammar that are not equivalent in their structure, making the grammar ambiguous. In our approach, we only consider *unambiguous* grammars for defining source and target languages, to ensure deterministic behaviour of the developed transformations (see Sec. IV for details).

Definition 3 (Unambiguous Grammar)**.** *A grammar* $G = (N, \Sigma, P, S)$ *is called* unambiguous *iff for every sentence* $s \in \mathcal{L}(G)$ *its derivation from* S *is unique if performed "leftmost derivation first".*

Because most models in MDE contexts are graph-based (since meta-models are graphs), we need context-free grammars producing graphs as sentences instead of strings. We use *hyperedge replacement graph grammars* (HR grammars) [8] which fulfil these requirements. HR grammars operate on *hypergraphs*, a generalization of graphs where edges, called *hyperedges*, can have more than two end points. These end points are called *attachment points* and their number is the arity of a hyperedge. Hyperedges take the role of non-terminal and terminal symbols. For replacing hyperedges in graphs by sub-graphs during rewriting, we need to specify how these sub-graphs are to be embedded. To this end, the replacing sub-graphs are equipped with *external nodes*, and rewriting proceeds by replacing the hyperedge with the sub-graph gluing together each external node with the attachment node of its corresponding attachment point.

More precisely, a *hyperedge replacement rule* $l_1 := H$ has three parts: a single n-ary non-terminal hyperedge l_1, a hypergraph H with $k \geq n$ external nodes replacing the hyperedge, and an (injective) *mapping* g of the k external nodes of H onto the n attachment points of l_1. Unlike string productions, graph replacement rules need to explicitly define how the new graph is attached to the remaining context. In HR grammars this is done via the mapping g. Fig. 3 shows a sketch of the replacement process (for $k = n$). First, the



(a) Matching (b) Removal (c) Gluing

Fig. 3: Rule application in hyperedge replacement grammars

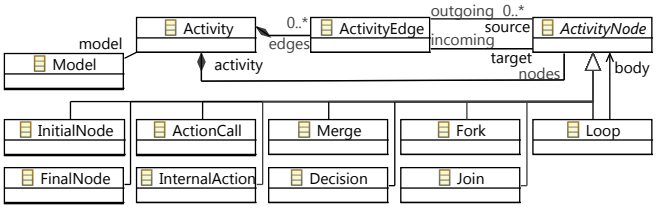


Fig. 4: Source meta-model: Activity diagrams

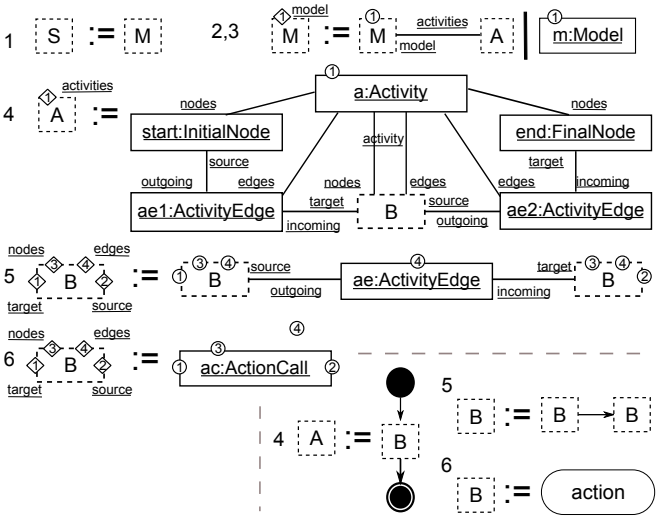


Fig. 5: HR productions subset for activity diagrams (abstract, concrete syntax)

hyperedge l_1 is found by matching (a) and removed (b), then the graph H is inserted (c) by gluing all its external nodes with the attachment nodes of l_1 according to the mapping g. Due to lack of space, we refrain from giving a more formal definition here. All definitions of string grammars carry over to HR grammars: an HR grammar has the same parts as a string grammar (non-terminals, terminals, productions and a start symbol) and its language is a set of hypergraphs. The membership problem for HR grammars is decidable [8] which guarantees the existence of the derivation trees we are going to use.

Now, we can define both the source and target language we use to demonstrate our grammar-based model transformation approach in terms of hyperedge replacement grammars. We exemplify our approach on a transformation from activity diagrams to the process algebra CSP [9]. The HR grammars for these two languages are compliant with the respective meta-models, i.e., the graphs which our grammars generate are all instances of the meta-models. For this, we use (a simplified version of) the meta-model of UML activity diagrams [11] (see Fig. 4) and the CSP meta-model from [12].

The meta-model of activity diagrams only contains basic diagram elements, their hierarchy, and associations with multiplicities. It does not describe higher-level *syntactic structures of the language*. For example, in a well-formed activity diagram each *decision node* should be eventually followed by the corresponding *merge node* for the branches of that decision. Although these kinds of inductive language structures are intuitive to the transformation developer, they are usually not described in the meta-model.

Fig. 5 shows six out of eleven productions of our source HR grammar. Productions are given in abstract (plus some in concrete) syntax, in the form $l := H$, using bars to distinguish different right-hand sides of productions. Non-terminal hyperedges are depicted by dashed lines. Types and the number of attachment points of a non-terminal hyperedge are determined by the associations of the meta-model elements which it groups. The mapping between attachment points and external nodes is depicted by using the same numbers, one given in a diamond and the other in a circle. Multiple external nodes can be mapped to one attachment point. Not all external nodes have to be connected in the graph (see production 6).

This grammar describes well-formed activity diagrams containing zero or more activities (productions 1 – 3) with exactly one *initial* and *final* node, and at most one recursively-defined high-level *syntactic structure block* called *B* connected to exactly one initial and final node (production 4). The non-terminal edge *B* can be replaced by one of the following structures: a *block sequence* (production 5), a *fork/join block*, a *decision/merge block*, a *loop*, an *internal action* or an *action call* (production 6). Note that our decision/merge constraint is now represented by the corresponding production which only allows to generate these elements together connected in a shape. Other language structures are also produced in this systematic way.

Fig. 6 shows the relevant subset of our meta-model compliant HR grammar for CSP. A *model* described by this grammar can contain a set of *processes* (productions 1 – 3) generated from non-terminal edges labelled *P*. A *process description* represented by non-terminal *PE* (production 4) can contain various expressions: a *sequential* or *parallel* process composition (productions 7 – 8), an *if-then-else* expression (production 9), an *event* followed by another process expression (production 10), *another process* (production 6), or an empty process *SKIP* (production 5).

B. Transformation Example

Both grammars we have defined will be used to describe our example transformation. We describe a transformation from activity diagrams to CSP, frequently employed for analysis purposes [13]. Alternatively, we could, for instance, use the activity diagrams to first-order logic transformation example from [14].

Fig. 7 shows our sample activity diagram of an enrolment and the corresponding CSP process. Here, we see the concrete

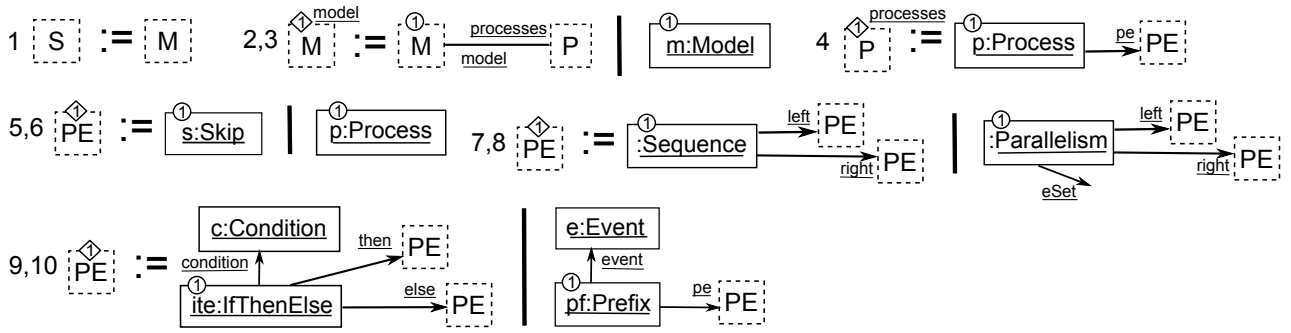


Fig. 6: HR productions subset for CSP (abstract syntax)

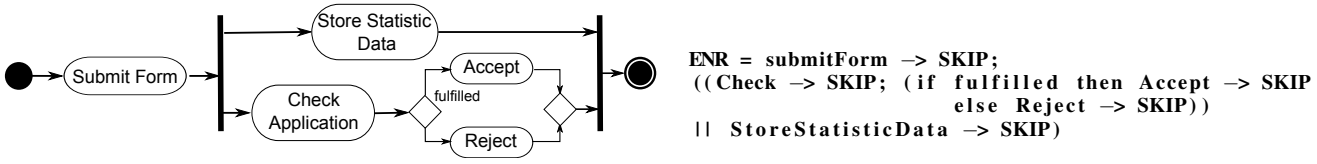


Fig. 7: Enrolment activity diagram (left) and its CSP process (`Check` abbreviates `CheckApplication` event) (right)

syntax of CSP: `->` is an event prefix, `;` a sequential composition, `SKIP` an empty process, `||` a parallel composition and `if-then-else` a conditional choice. We require every activity to be transformed into a process and every block sequence and fork/join block into a sequential and parallel process composition, respectively. A decision/merge block is to be transformed into an if-then-else expression and an action call into an event followed by a SKIP. Loops, although absent in this example, are transformed into recursive processes with conditions. Next, we show how this transformation logic can be structurally described using our approach.

III. Transformation Development

Our main goal is to allow an intuitive model transformation definition by mapping high-level syntactic structures in source and target languages onto each other. In terms of grammars, this means relating (or *pairing*) source and target productions creating corresponding structures. During execution of the model transformation these relations will be used to identify which target production is triggered for which source production.

Our model transformations operate on derivation trees of source and target models. Given a derivation tree for a source model, we incrementally construct a derivation tree for the target model by applying corresponding productions. Thereby, 1-to-1 correspondences between non-terminal edges in the source and target derivation trees help to keep track of related model structures.

Fig. 8 shows a sample transformation rule relating the source production for a sequence of activity blocks to the target production for a sequence of process expressions[1]. It states that when a block sequence is replacing a non-terminal hyperedge

[1]To simplify, we show transformation rules without HR grammar details.

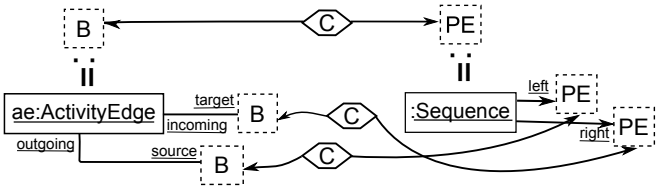


Fig. 8: Transformation rule: Block sequence to process sequence

of type *B* (block) in a source derivation tree, a sequence of process expressions should replace the corresponding hyperedge of type *PE* (process expression) in the related target derivation tree. In addition to relating productions, the transformation rule links source non-terminal edges of type *B* and target non-terminal edges of type *PE* via a 1-to-1 correspondence of type *C*. These correspondences determine which target edge will be replaced by the target production, when the linked source edge is replaced by the related source production. The notion of correspondence is inspired by triple graph grammars (TGGs) [15].

Fig. 9 shows further rules of the transformation definition for our example. It relates productions in the following way:

- Rules 1, 2: Productions for non-terminal (1) and terminal (2) edges representing *models* in the source and target grammars are related. Non-terminal edges in rule 1 are linked via a correspondence later required by rules 2 – 3.
- Rule 3: Production of a non-terminal edge of type *A* representing an *activity* is related to the production of a non-terminal edge of type *P* representing a *process*. The correspondence between model non-terminal edges is kept and other produced edges are linked for later application of rule 4.
- Rule 4: The production of an activity containing a *block* represented by a non-terminal edge of type *B* connected

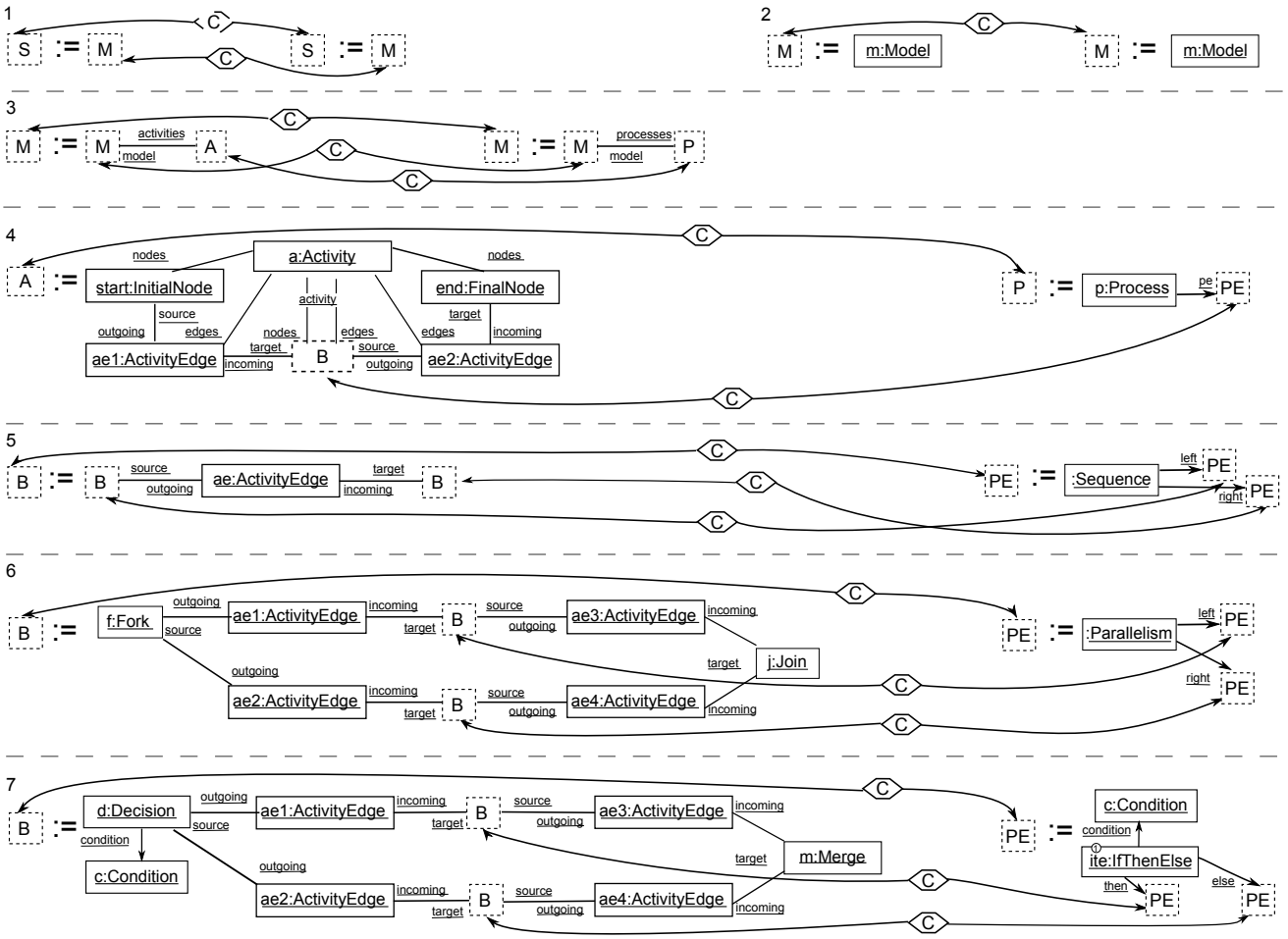


Fig. 9: Transformation definition (fragment): Activity diagram to CSP

to the initial and final nodes is related to the production of a process with a *process expression* represented by the non-terminal edge of type *PE*. The produced non-terminal edges are linked via correspondences which are required later for applying rules 5 – 7.

- Rules 5 – 7 relate different types of activity blocks to different process expressions: A *sequence of blocks* is related to a *sequence of process expressions* (5), a *fork/join* block to a *parallel composition* of process expressions (6), a *decision/merge* block to an *if-then-else* expression (7). Correspondences are created on the same principle as before.

In general, if the transformation definition between two HR grammars G_s and G_t has the following form (extended in Sec. V):

1) In each rule:
   a) a source production $p_s = (n_s, r_s)$ is related to a target production $p_t = (n_t, r_t)$;
   b) left-hand side non-terminals n_s and n_t are linked via a correspondence (start non-terminals S_s, S_t are always linked);
   c) each non-terminal edge in r_t has exactly one corresponding non-terminal edge in r_s,
2) For each source production $p_s = (n_a, r_a)$ in P_s and each correspondence between the edges of type n_a and n_b in some rule (or initial S_s to S_t correspondence), there is exactly one rule relating p_s to a target production p_t, where $p_t = (n_b, r_t)$, to cover all combinations of types of corresponding pairs (n_a, n_b),

and G_s is unambiguous, then the resulting transformation has some important characteristics which we show in Sec. IV. Now, we discuss how the transformation rules we have just defined are executed.

The transformation is executed on a source model in two steps: first, the source model is parsed to get its *leftmost derivation tree*[2] and then the transformation rules are applied to construct the target derivation tree (and the target model).

In the first step, a source model is parsed with respect to the source HR grammar. As HR grammar based parsing is decidable [8], for each source model we either get its leftmost derivation tree or a message that it is not parsable. If a model can not be parsed, it is not in the source language, and hence will be rejected by our transformation returning *not applicable*.

[2]Derivation tree representing leftmost derivation. Leftmost derivation in HR grammars is analogous to the one for string grammars (see [7]).

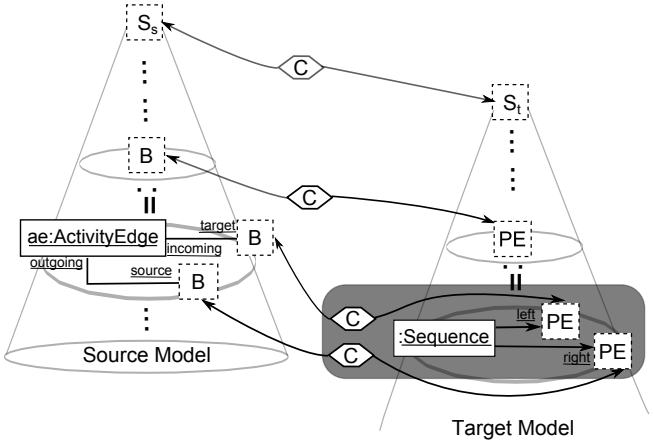


Fig. 10: Transformation execution sketch

In the second step, we build the target derivation tree by first initializing it by the edge of type S_t which has correspondence to the edge of type S_s in the source tree. Next, we iteratively construct the target tree in the following way: we traverse the source tree to find the next non-terminal edge e_s and the source production p_s that has rewritten it. Then, we consider each correspondence c of the edge e_s and find the transformation rule that pairs some p_t with the source production p_s, and where the left-hand side non-terminals are equal to the types of the edges linked by c. Finally, we apply the target production p_t to the target non-terminal edge linked to e_s through c, and create additional correspondences according to the transformation rule. The transformation terminates once the complete source derivation tree has been traversed and all correspondences have been considered. Due to the context-freeness of HR grammars [8], we can use any traversal method.

Fig. 10 sketches the transformation execution, highlighting a single production in the source derivation tree for our example applied to an edge of type *B* with the corresponding edge *PE* in the already created target tree fragment. The dark grey rectangle frames the result of applying the suitable transformation rule 5 (Fig. 9) to the corresponding edge *PE*.

IV. Transformation Properties

Besides allowing for a natural way of transformation definition by mapping logically equivalent concepts onto each other, our method for building model transformations exhibits some important properties. A transformation defined using our approach and fulfilling the criteria mentioned above is, by construction:

a) *terminating* – for any input model, the transformation terminates and returns either a resulting model or *not applicable*,
b) *complete* – all valid, i.e., parsable, models are transformable,
c) *sound* – a valid and transformable model is always transformed into a valid model (parsable in the target grammar),
d) *deterministic* – the output model and its derivation are fully determined by the input model.

For the last property, we require the source grammar to be unambiguous (see Def. 3). Due to the page limit, we only give proof sketches for each of the properties here. In the sketches, we emphasize the conditions on the transformation definition that are sufficient to guarantee these properties. In the following we refer to the source and target HR grammars as G_s and G_t, to the input (source) model as M_s, to the output (target) model as M_t, and to the transformation as τ.

*a) **Termination**:* As described in the last section, τ first parses the input model M_s, yielding a source model leftmost derivation tree T_s (or *not applicable*). Since our approach is based on *context-free* grammars, this is guaranteed to terminate. Then, a *single* G_t production applications is performed by τ for every G_s production application and correspondence c of the edge rewritten by it in T_s. Since these productions' applications are guaranteed to terminate, and the set of correspondences, and T_s *are finite*, the whole process is also guaranteed to terminate.

*b) **Completeness**:* For completeness, we have to show that if $M_s \in \mathcal{L}(G_s)$, $\tau(M_s)$ will not fail. If $M_s \in \mathcal{L}(G_s)$, the first step (parsing) will always succeed, and return T_s. In Sec. III, we have demanded that τ contains a transformation rule for *every production* in G_s and *every correspondence type* an edge rewritten by it might have. Hence, we can transform every production application in T_s into an application in T_t.

Next, we look at the derivation of the output target model M_t via T_t. For completeness, we need to show that this derivation does not fail, i.e., that all target productions are applicable at the place where the transformation wants to apply them. Both G_s and G_t are *context-free*, so the existence of the non-terminal edges ensured by the correspondences is all that is needed to ensure applicability of the target productions. When considering the next source production application, we apply the related target production to the non-terminal edge corresponding to the edge rewritten by the source production. Therefore, since the source production is applicable and τ contains rules for every type of correspondence that non-terminal edge rewritten by it might have, the target production is applicable too. Thus, τ always returns a model for a valid M_s and thus, τ is complete.

*c) **Soundness**:* It remains to be shown that $M_t \in \mathcal{L}(G_t)$. This can be reduced to the question whether the target tree T_t is complete, meaning that no non-rewritten non-terminal edges are left after the application of the transformation.

From requirements in Sec. III, every non-terminal edge produced by a target production is *linked by a correspondence* to a non-terminal edge in T_s. And since $M_s \in \mathcal{L}(G_s)$, all source non-terminal edges produced are eventually rewritten. This implies that all target non-terminal edges produced are also eventually rewritten by the related target productions, i.e., T_t is complete. Thus, we have $M_t \in \mathcal{L}(G_t)$.

*d) **Determinism**:* Since G_s *is unambiguous*, the production of T_s is deterministic. The tree T_s fully determines which rules are applied to construct the target tree, and where. This is because each production on the target side is uniquely determined by the source production and the correspondence

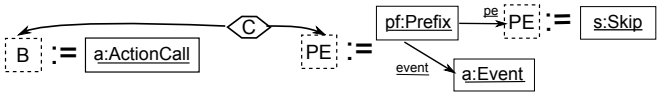


Fig. 11: Transformation rule: Action to an event followed by SKIP process

of the source non-terminal edge it rewrites. The target side of this correspondence also uniquely determines where the target production is applied. We thus, obtain exactly one target derivation tree, and consequently exactly one target model. Therefore, no two different target models can be the transformation result for a single M_s.

V. Extensions

As you might have noticed, we still have to describe a couple of rules to complete the definition of our sample transformation. The remaining rules need to transform: an action call (1) or an internal action (2) into an event followed by a SKIP process, and a loop (3) into an equivalent recursive process, since CSP does not natively support loops. In these two cases we need to relate *two or more* target productions that form a derivation sub-tree to a single source production. To allow this, we extend our approach by relating *derivation sub-trees* instead of relating *single productions* in transformation rules.

Fig. 11 shows such an extended rule for case 1 (2 is analogous): it relates the production for an action call to a sub-tree combining two target productions to create an event followed by a SKIP process. Since all non-terminal edges created by these target productions (one *PE*) have been consumed in the sub-tree, we only require a correspondence for left-hand side non-terminals of the first productions.

Fig. 12 shows the rule for case 3 and, thereby, the second extension of our approach – so called *non-local rules*. This type of rule allows the use of an *additional correspondence* between non-terminals, and uses its target non-terminal as a root of an *additional derivation sub-tree* containing one or more productions. During the transformation execution, when a correspondence of the defined additional type is found and the rule can be applied, the defined additional sub-tree is assigned to its target of this correspondence, replacing it.

Both extensions can be combined within one rule as seen in Fig. 12: it relates the production for a loop to the production for a process reference *L(c)* and uses a correspondence between two model non-terminals (*M*) to create the referenced process *L(x)*. This process consists of an *if-then-else* expression with the first non-terminal edge *PE* in the *then* branch linked to the body of the activity loop represented by the edge *B*, followed by the recursive call to itself, which terminates when the condition *x* fails. The new process is placed directly under the output model edge *M* which makes the result of this rule *non-local* and different from previous examples.

As for the conditions we imposed on our transformation for it to have the desired properties (see Sec. IV), they now need to be adjusted. A full explanation is out of the scope of this paper, thus, we only provide an idea of required adjustments.

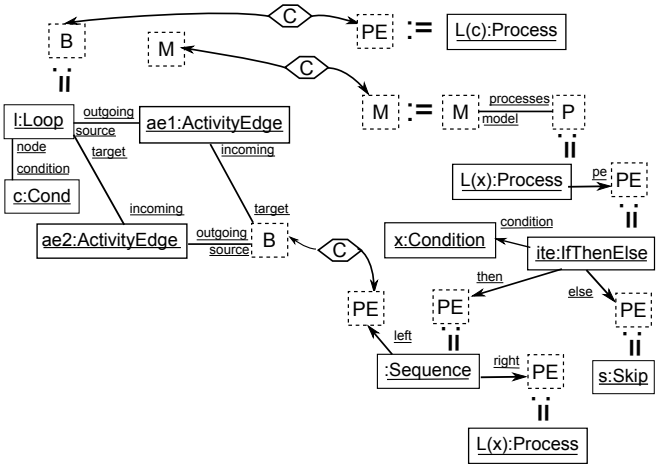


Fig. 12: Transformation rule: Loop to an equivalent recursive process

First, for multiple properties, when a rule relates derivation sub-trees, all non-terminal leaves of the target sub-tree have to have corresponding non-terminal leaves in the source sub-tree. Next, for determinism, the absence of two simultaneously applicable transformation rules need to be guaranteed: either by disallowing rules with the same correspondence and one source sub-tree being a sub-tree of the other, or by introducing rule priorities. Furthermore, for completeness, the presence of rules covering all possible source model derivation steps must be guaranteed: either via coverage analysis or forbidding to use source sub-trees with more than one production. In the case of non-local rules, to guarantee determinism, exactly one correspondence of the specified additional type must exist when the rest of the rule is applicable. Plus, the same correspondence condition as for the main target sub-tree must hold for non-terminal leaves of the additional target sub-tree.

VI. Tool Support & Evaluation

To support the approach we have designed a tool chain involving existing state-of-the-art tools: a graph grammar parser of the AGG [16] framework and the EMorF [17] transformation engine. The chain requires source and target grammars modelled as instances of our HR grammar meta-model extended with grammar-specific non-terminal and terminal type classes. All meta-models are specified in the *ECore* meta-modelling language of the *Eclipse Modelling Framework* [18]. The specified source and target grammars are used as input of our *Eclipse* based *Grammar-Based Model Transformation Framework* that allows the definition of transformation rules and validate their compliance with the conditions from Sec. III. The framework makes use of existing transformation languages and engines like TGGs [15] and EMorF engine by translating the created transformation into its declarative, language (and engine) specific implementation operating on trees. Other combinations of languages and engines like ATL [19] with its engine can also be used.

Fig. 13 shows the described tool chain with the used artefacts for the case when TGGs and the TGG engine (here

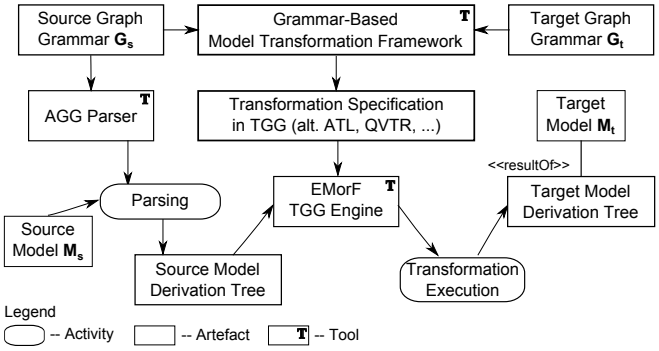


Fig. 13: Tool support for grammar-based transformations with TGGs and EMorF as implementation platform

EMorF) were chosen as the target execution platform for the developed transformation.

To evaluate our approach, we compare it on our running example with the most common transformation development practices in MDE, where meta-models are used in combination with declarative, imperative, and hybrid transformation languages. Meta-models may contain additional classes (e.g., *StructuralActivityNode* in UML [11]) to group (other) classes.

When a source meta-model does not include such structural classes, then the transformations that require this structural information defined on it have to use *imperative constructs*. Often they are also combined with recursion. In our example, such constructs are required to locate the related decision and merge activity nodes to build the corresponding sequence of processes in CSP.

```
rule Decision2IfThenElse extends DefaultNode2Skip {
from d  : AD!Decision
to   ps : ProcessExpression!Sequence (
   left  <- pe,
   right <- d.findMrg(d.outgoing.first().target,0)
 ),

 pe : ProcessExpression!IfThenElse (
   then <- d.outgoing->select(...).first().target,
   else <- ..., condition <- ...
)}

helper def:findMrg(n:AD!ActivityNode,i:Integer)
: AD!Merge =
if n.oclIsTypeOf(AD!Merge) then
 if i > 0 then
  thisModule.findMrg(n.outgoing.first().target,i-1)
 else
  n
 endif
else
 if n.oclIsTypeOf(AD!Decision) then
  thisModule.findMrg(n.outgoing.first().target,i+1)
 else
  thisModule.findMrg(n.outgoing.first().target,i)
 endif
endif;
```

Listing 1: ATL code fragment: Decision / merge block to if-then-else process sequence

Listing 1 shows implementation of this transformation step in a widely used general-purpose hybrid model transformation language ATL [19], which relies on meta-model based language definition. We use the meta-models from Sec. II-A to define the source and target languages of the ATL transformation. We use an ATL matching rule to transform a decision node d to an if-then-else process expression pe (rule *Decision2IfThenElse*). But to link the target expression pe to the next process expression, which should be the result of transforming the merge node corresponding to the decision node d, we have to use recursion to find that merge node. For this purpose, we implement a recursive search helper *findMrg* that follows the path starting in d and skips intermediate decision/merge pairs until it finds the right merge node.

To use this recursive search in ATL, we have to assume well-formedness of activity diagrams, ensured by extra OCL constraints. Such OCL constraints are not always defined and, if they are, they contain recursion and considerably complicate the complete meta-model based language definition. Furthermore, the need for imperative constructs forbids the use of declarative languages and significantly complicates readability and analysis of the transformations. In fact, most techniques aiming to guarantee transformation quality [20], [21] only consider declarative rules and are not applicable here.

Another solution to build structure-based transformations is to use transformation rules to create the required structures in a suitably defined (by meta-model containing structural classes) correspondence model as done in [22] using TGGs. As the previous imperative solution, the correspondence-based solution does not guarantee quality of the developed transformations, and it complicates their understandability and analysis.

If meta-models include structural classes, it is possible to define declarative structure-based rules in TGG-like format with better readability than in the previous solutions, and with more analysis possibilities. Still, as far as we know, there are no approaches showing completeness and determinism even for such declarative transformations. Unlike these common practices, our approach natively supports structure-based transformations keeping their rules graphical and concise (see Fig. 9, rule 7), and guarantees their quality.

Discussion: Transformation rules defined using our approach stay declarative (see Fig. 9) which brings multiple advantages: simpler and more intuitive transformation rules that are easier to understand and maintain especially when grammar productions are represented in concrete syntax; and guaranteed transformation quality properties discussed in Section IV. The only part that stays imperative is the source model parsing. Parsing causes most of the complexity in our method, but its mathematical foundations for HR grammars [8] guarantee termination and predictable worst-case run-time.

The conditions we currently put on transformations defined by our approach to guarantee their quality (Sec. III) might be too restrictive even with the use of extensions (Sec. V) for some purposes. In such cases, a developer can still define structure-based declarative transformations using our approach and employ existing testing and verification techniques to check their quality.

The choice of HR grammars for language definition currently limits our approach to the transformations between

context-free graph-based languages. We plan to address this limitation and take context-sensitive languages into consideration using grammars proposed in [23].

HR graph grammars, which we use to define source and target languages, in general, are more restrictive and complex than pure meta-models (with structural classes). However, when compared to meta-models with OCL constraints enforcing (when possible) the same structural well-formedness constraints on e.g., activity diagrams, HR grammars typed over meta-models present a more concise, intuitive, and powerful way to describe such constraints.

An a meta-model, an HR grammar only needs to be created once per language and, can then be used by any developer for any grammar-based transformation involving this language. The complexity of an HR grammar can affect the complexity of the grammar-based transformation using it, but this is also the case with meta-models.

VII. Related Work

For years, compiler construction benefits from syntax-directed translation [6]. This technique relates single string grammar productions via 1-to-1 relations and requires both grammars to have the same non-terminals, building a very basic version of our approach. Pratt [7] was first to propose to apply this technique to graphs and show that the resulting transformations are deterministic and reversible (under conditions). Our approach extends [7] to n-to-m relations between productions, n-to-m correspondences between non-terminals, and shows additional properties for the developed transformations. Thus, we consider a much larger scope of transformations than the approach from [7].

TGGs proposed in [15] were also inspired by Pratt [7] and were first to contain explicit correspondence nodes. However, the focus of TGGs was on relating context-sensitive productions to support data integration without the consideration of transformation quality properties. In MDE today, TGGs are defined on meta-models and relate model patterns gradually matched during the execution instead of grammar productions. The only approach we are aware of, that uses TGGs with meta-models to define structure based transformation in [22], has already been compared to in out evaluation in Sec. VI.

Halfway between grammars, as in our approach, and meta-models is the transformation development and validation approach proposed in [24]: it relates source model patterns with target productions and states extensive criteria for the transformation quality assurance. Target language in this approach is defined through graph grammar productions. However, it is unclear how the target grammars used there are defined and whether the related target productions can always be applied when the source pattern is found during the transformation execution. In [24] transformation execution strategy is defined manually whereas, in our approach it is automatically obtained during the source model parsing making it less error prone. Like us, authors of [24], consider transformation characteristics: termination, and confluence; however, they do not consider soundness and completeness.

Other approaches like [25], [26], [27] advocate transformation development by-example and by-demonstration, and do not directly focus on structure based transformations. Like us, these approaches recognize the problem of over abstraction of language definition through meta-models, but deal with it using examples to describe corresponding structures of the source and target languages, whereas we use graph grammar productions. By such example-based approaches it is not always clear whether the examples or the transformation should be adapted when the result is not yet satisfactory, how many examples are needed. Whether the developed transformation has desired properties is, as far as we know, not addressed by any of these approaches.

Several other approaches [28], [29] directly apply classic techniques to model transformations. Unfortunately, non of them considers properties of the developed transformations.

In [28] the authors attempt to use TXL [30] – a generic source transformation framework – to develop model transformations. They consider meta-model based languages, and transform them into TXL string grammars. TXL string grammars do not have the expressiveness and visualization advantages of graph grammars we use leading to very limited applicability of the TXL-based approach. Transformations described in TXL are fine-grained with explicit execution policy, which makes them flexible, but also complex and difficult to understand and maintain. This method can be placed halfway between the syntax-directed translation and our approach.

In [29] the authors attempt to simplify transformation development by eliminating the need to learn specialized languages. They regard models and meta-models as abstract data types – abstract structures with operations. On top of the types they define a minimal imperative model transformation language with formal semantics. This approach brings models and transformations into the world of programming, whereas our approach lifts translation techniques to graphs. Both last approaches use meta-models and none of them directly considers high-level structures in languages and transformations based on these structures, as we do. Transformation quality is not take under consideration either.

We consider our approach as the next step towards efficient and quality-aware transformation development that can be realized based on existing state-of-the-art including some approaches described above and the commonly used technologies like ATL [19] and TGG [15].

In general, use of alternative notations for modelling language definition – meta-model vs. graph grammar – raises the issue of integration and interoperability of approaches and tools respectively. Various methods address this issue by defining transformations between the alternatives [31], [32], [33], applying inference to obtain graph grammars [34], or combining them as different views for multi-level modelling [35]. The later option is the one we use.

Finally, we want to point out that the simplest version of our method has been recently successfully used in [36] for semantic-based machine translation in the field of computational linguistics.

VIII. Conclusion

In this paper, we have presented a grammar-based model transformation development approach that allows to naturally consider structures of involved language. We have employed HR grammars to specify source and target languages, and defined transformation rules by relating their productions and adding correspondences between non-terminals. We have shown that model transformations defined using our approach terminate and are sound, complete, and deterministic. We have also presented some extensions of the approach.

Future Work: Currently, we are working on the extension of initial case studies to evaluate our approach and continue improving the tool support. In the future, we look to support computation of attributes, while still keeping the desired transformation properties. When expressiveness of HR grammars is not sufficient, we plan to explore the decidable contextual graph grammars proposed by Drewes in [23].

References

[1] K. Czarnecki and S. Helsen, "Feature-Based Survey of Model Transformation Approaches," *IBM Systems Journal*, vol. 45, no. 3, pp. 621–646, 2006. doi: 10.1147/sj.453.0621

[2] D. D. Ruscio, R. Eramo, and A. Pierantonio, "Model Transformations," in *SFM*, ser. LNCS, M. Bernardo, V. Cortellessa, and A. Pierantonio, Eds., vol. 7320. Springer, 2012. doi: 10.1007/978-3-642-30982-3_4 pp. 91–136.

[3] "Meta Object Facility (MOF) Core Specification." [Online]. Available: http://www.omg.org/spec/MOF/

[4] "Object Constraint Language (OCL)." [Online]. Available: http://www.omg.org/spec/OCL/

[5] "Extended BNF," ISO/IEC 14977, Int. Organization for Standardization, 2001.

[6] A. Aho, M. Lam, R. Sethi, and J. Ullman, *Compilers: Principles, Techniques, and Tools*. Pearson/Addison Wesley, 2007. ISBN 0-321-48681-1

[7] T. W. Pratt, "Pair Grammars, Graph Languages and String-to-Graph Translations," *Journal of Computer and System Sciences*, vol. 5, no. 6, pp. 560 – 595, 1971. doi: 10.1016/S0022-0000(71)80016-8

[8] G. Rozenberg, Ed., *Handbook of Graph Grammars and Computing by Graph Transformation*. World Scientific Publishing Co., Inc., 1997, vol. 1. ISBN 98-102288-48

[9] C. A. R. Hoare, *Communicating sequential processes*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1985. ISBN 0-13-153271-5

[10] E. Syriani and J. Gray, "Challenges for Addressing Quality Factors in Model Transformation," in *ICST*, G. Antoniol, A. Bertolino, and Y. Labiche, Eds. IEEE, 2012. doi: 10.1109/ICST.2012.198 pp. 929–937.

[11] "Unified Modeling Language (UML)." [Online]. Available: http://www.omg.org/spec/UML/

[12] D. Varró, M. Asztalos, D. Bisztray, A. Boronat, D.-H. Dang, R. Geiß, J. Greenyer, P. V. Gorp, O. Kniemeyer, A. Narayanan, E. Rencis, and E. Weinell, "Transformation of UML Models to CSP: A Case Study for Graph Transformation Tools," in *AGTIVE*, ser. LNCS, A. Schürr, M. Nagl, and A. Zündorf, Eds., vol. 5088. Springer, 2007. doi: 10.1007/978-3-540-89020-1_36 pp. 540–565.

[13] G. Besova, S. Walther, H. Wehrheim, and S. Becker, "Weaving-Based Configuration and Modular Transformation of Multi-layer Systems," in *MoDELS*, ser. LNCS, R. B. France, J. Kazmeier, R. Breu, and C. Atkinson, Eds., vol. 7590. Springer, 2012. doi: 10.1007/978-3-642-33666-9_49 pp. 776–792.

[14] S. Walther and H. Wehrheim, "Knowledge-Based Verification of Service Compositions – An SMT Approach," in *Engineering of Complex Computer Systems (ICECCS), 2013 18th International Conference on*, July 2013. doi: 10.1109/ICECCS.2013.14 pp. 24–32.

[15] A. Schürr, "Specification of Graph Translators with Triple Graph Grammars," in *WG*, ser. LNCS, E. W. Mayr, G. Schmidt, and G. Tinhofer, Eds., vol. 903. Springer, 1994. doi: 10.1007/3-540-59071-4_45 pp. 151–163.

[16] G. Taentzer, "AGG: A Tool Environment for Algebraic Graph Transformation," in *AGTIVE*, ser. LNCS, M. Nagl, A. Schürr, and M. Münch, Eds., vol. 1779. Springer, 2000. doi: 10.1007/3-540-45104-8_41 pp. 481–488.

[17] L. Klassen and R. Wagner, "EMorF - A tool for model transformations," *ECEASST*, vol. 54, 2012.

[18] "Eclipse Modeling Framework (EMF)." [Online]. Available: http://www.eclipse.org/modeling/emf

[19] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, "ATL: A model transformation tool," *Science of Computer Programming*, vol. 72, no. 1-2, pp. 31–39, 2008. doi: 10.1016/j.scico.2007.08.002

[20] J. Cabot, R. Clarisó, E. Guerra, and J. de Lara, "Verification and Validation of Declarative Model-to-Model Transformations Through Invariants," *J. Syst. Softw.*, vol. 83, no. 2, pp. 283–302, 2010. doi: 10.1016/j.jss.2009.08.012

[21] F. Büttner, M. Egea, J. Cabot, and M. Gogolla, "Verification of ATL Transformations Using Transformation Models and Model Finders," in *ICFEM*, ser. LNCS, T. Aoki and K. Taguchi, Eds., vol. 7635. Springer, 2012. doi: 10.1007/978-3-642-34281-3_16 pp. 198–213.

[22] C. Lohmann, J. Greenyer, J. Jiang, and T. Systä, "Applying Triple Graph Grammars For Pattern-Based Workflow Model Transformations," *Journal of Object Technology*, vol. 6, no. 9, pp. 253–273, 2007. doi: 10.5381/jot.2007.6.9.a13

[23] F. Drewes, B. Hoffmann, and M. Minas, "Contextual Hyperedge Replacement," in *AGTIVE*, ser. LNCS, A. Schürr, D. Varró, and G. Varró, Eds., vol. 7233. Springer, 2012. doi: 10.1007/978-3-642-34176-2_16 pp. 182–197.

[24] J. M. Küster, "Definition and validation of model transformations," *Software and Systems Modeling*, vol. 5, pp. 233–259, 2006. doi: 10.1007/s10270-006-0018-8

[25] D. Varró, "Model Transformation by Example," in *MoDELS*, ser. LNCS, O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, Eds., vol. 4199. Springer, 2006. doi: 10.1007/11880240_29 pp. 410–424.

[26] P. Langer, M. Wimmer, and G. Kappel, "Model-to-Model Transformations By Demonstration," in *ICMT*, ser. LNCS, L. Tratt and M. Gogolla, Eds., vol. 6142. Springer, 2010. doi: 10.1007/978-3-642-13688-7_11 pp. 153–167.

[27] Y. Sun, J. White, and J. Gray, "Model Transformation by Demonstration," in *MoDELS*, ser. LNCS, A. Schürr and B. Selic, Eds., vol. 5795. Springer, 2009. doi: 10.1007/978-3-642-04425-0_58 pp. 712–726.

[28] H. Liang and J. Dingel, "A Practical Evaluation of Using TXL for Model Transformation," in *SLE*, ser. LNCS, D. Gašević, R. Lämmel, and E. Wyk, Eds., vol. 5452. Springer, 2009. doi: 10.1007/978-3-642-00434-6_16 pp. 245–264.

[29] J. Irazábal and C. Pons, "Model Transformation Languages Relying on Models as ADTs," in *SLE*, ser. LNCS, M. Brand, D. Gašević, and J. Gray, Eds., vol. 5969. Springer, 2010. doi: 10.1007/978-3-642-12107-4_10 pp. 133–143.

[30] J. R. Cordy, "The TXL source transformation language," *Sci. Comput. Program.*, vol. 61, no. 3, pp. 190–210, 2006. doi: 10.1016/j.scico.2006.04.002

[31] M. Wimmer and G. Kramler, "Bridging Grammarware and Modelware," in *Satellite Events at the MoDELS*, ser. LNCS, J.-M. Bruel, Ed., vol. 3844. Springer, 2006. doi: 10.1007/11663430_17 pp. 159–168.

[32] B. Hoffmann and M. Minas, "Generating Instance Graphs from Class Diagrams with Adaptive Star Grammars," *ECEASST*, vol. 39, 2011.

[33] B. Henderson-Sellers, "Bridging metamodels and ontologies in software engineering," *Journal of Systems and Software*, vol. 84, no. 2, pp. 301–313, 2011. doi: 10.1016/j.jss.2010.10.025

[34] A. Stevenson and J. R. Cordy, "Grammatical Inference in Software Engineering: An Overview of the State of the Art," in *SLE*, ser. LNCS, K. Czarnecki and G. Hedin, Eds., vol. 7745. Springer, 2012. doi: 10.1007/978-3-642-36089-3_12 pp. 204–223.

[35] C. Atkinson, R. Gerbig, and C. Tunjic, "Towards Multi-level Aware Model Transformations," in *ICMT*, ser. LNCS, Z. Hu and J. de Lara, Eds., vol. 7307. Springer, 2012. doi: 10.1007/978-3-642-30476-7_14 pp. 208–223.

[36] B. Jones, J. Andreas, D. Bauer, K. M. Hermann, and K. Knight, "Semantics-Based Machine Translation with Hyperedge Replacement Grammars," in *COLING*, M. Kay and C. Boitet, Eds. Indian Institute of Technology Bombay, 2012, pp. 1359–1376.