On validation of ATL transformation rules by transformation models

Fabian Buettner, Jordi Cabot, Martin Gogolla

To cite this version:

Fabian Buettner, Jordi Cabot, Martin Gogolla. On validation of ATL transformation rules by transformation models. MoDeVVa Proceedings of the 8th International Workshop on Model-Driven Engineering, Verification and Validation, Oct 2011, Wellington, New Zealand. hal-00691062

HAL Id: hal-00691062
https://inria.hal.science/hal-00691062
Submitted on 25 Apr 2012
ABSTRACT
Model-to-model transformations constitute an important ingredient in model-driven engineering. As real world transformations are complex, systematic approaches are required to ensure their correctness. The ATLAS Transformation Language (ATL) is a mature transformation language which has been successfully applied in several areas. However, the executable nature of ATL is a barrier for the validation of transformations. In contrast, transformation models provide an integrated structural description of the source and target metamodels and the transformation between them. While not being executable, transformation models are well-suited for analysis and verification of transformation properties. In this paper, we discuss (a) how ATL transformations can be translated into equivalent transformation models and (b) illustrate how these surrogates can be employed to validate properties of the original transformation.

Categories and Subject Descriptors
D.2.4 [Software Engineering]: Software/Program Verification—Validation

General Terms Languages, verification

Keywords
Validation of model transformations, ATL, transformation models

1. MOTIVATION
Model-driven engineering has been acknowledged as an effective way to tackle the complexity of software and systems engineering. Model-to-model (M2M) transformation is an important ingredient in model-driven settings. There exist several approaches to describe this kind of transformations, operational as well as declarative ones. While declarative approaches describe the relationship between the source and the target model, operational approaches describe the execution steps for the creation of the target model from the source model.

Irrespective of the formalism, real world transformations are complex. In order to provide reliable quality transformations, systematic approaches are required to ensure their correctness. To address this problem, several approaches have been developed in two categories: (a) formal verification and (b) validation of model transformations by means of systematic (bounded) testing. For both kinds, translations of model transformation languages into other formalisms such as rewriting logic, higher-order logic, and model consistency checking languages have been developed.

In [6, 17] we introduced the notion of a transformation model as a declarative and integrated way to specify M2M transformations. In short, a transformation model is a unified structural description of the source metamodel, the target metamodel, and the relationship between them that is established by a transformation. Unlike other declarative approaches to model transformations, transformation models do not introduce new formalisms, the ‘usual’ meta-modeling ingredients (MOF/Ecore/KM3 plus OCL constraints) are sufficient to describe the transformation model. In general, transformation models are not directly executable. However, due to their integrated structural character, they are well-suited to investigate several properties of model transformations.

The ATLAS Transformation Language (ATL) [20] is a mature M2M transformation language that is aligned with MOF/QVT [25]. ATL has been applied in several areas, [29] alone lists 103 different transformations in the ATL ‘transformation zoo’. However, the executable nature of ATL provides a barrier for the direct investigation of transformation properties.

In this paper, we propose an approach for the validation of ATL transformations by means of generated transformation models. An important contribution of our approach is that it provides an equivalent integrated representation of the precise syntax of the source and target models (i.e., including their OCL well-formedness constraints) and the execution semantics of the ATL transformation. Thus, while we still formulate the transformations using ATL first, we gain the benefits of an equivalent transformation model for validation purposes. Inherently, our approach provides an intuitive means to express the trace between the source and the target, making it accessible for investigation as well. It comprises two core elements:

1. A translation of ATL transformations into structural transformation models (employing OCL constraints to express the execution semantics of ATL). In this paper we provide a first approach towards a systematic translation. The development of a complete and universally valid translation scheme is ongoing work.

2. We show how to use the transformation model for validation purposes by employing UML validation and verification methods. In particular, we rely on approaches such as
[11], [1], and [18] to automatically validate the transformations. As an example, we investigate the important question whether a transformation may produce invalid target models from valid source models, as this indicates a weakness in the transformation that probably should be fixed. In general, several other properties of model transformations can be analyzed in a similar fashion.

The remaining paper is structured as follows: In Sect. 2 we provide a simple ATL model transformation ER2REL from an Entity-Relationship (ER) to a relational (REL) metamodel as an example to illustrate our approach. In Sect. 3 we explain how the ATL rules can be translated systematically into a transformation model. Section 4 shows how this transformation model can be exploited to validate ER2REL, identifying several bugs in ER2REL by bounded model consistency checking. Section 5 puts our approach in the context of related work. We conclude in Sect. 6 and explain further research objectives.

2. EXAMPLE

In this section we first introduce two simple metamodels ER and REL that we will act as source and target for a model transformation ER2REL. We chose these domains for our example because they have well-known semantics and concrete syntax, and can be formulated as metamodels small enough to fit in a short paper. However, as we will show later in Sect. 4, the transformation leaves room for several pitfalls.

2.1 ER and REL Metamodels

Both the ER and the REL metamodel can be described using metatamodels such as MOF, Ecore, and KM3, following the usual metatmodel architecture. Which concrete metamodel is chosen is not relevant for the subsequent. Figure 1 shows the class diagrams for the metamodels. For the sake of simplicity we omitted attribute types and relationship multiplicities in our metamodels. Furthermore, we simply refer to the elements of a ER database schema as entities and relationships instead of entity types and relationship types (which would be important if we also dealt with instances of the database schemas).

![Figure 1: ER and REL metamodels](image_url)

In addition to the class diagrams of ER and REL, OCL invariant constraints describe the well-formedness of both metamodels as shown in Listings 1 and 2. For ER, we require that entity names and relationship names are unique within a database schema (constraints ER.EN and ER.RN). Analogously, attribute names have to be unique within entities and relationships (ER.EN and ER.RN). Attributes of relationships are not allowed to have key attributes (ER.K), as the identity of a relationship instance should be determined by the participating entity instances. For REL, we have only three constraints: Relation names have to be unique within a relational database schema (REL.RN), attribute names have to be unique within a relation (REL.RAN), and every relation has to have at least one key attribute (REL.K).

![Listing 1: Constraints of ER](image_url)

For the sake of readability, we will use concrete syntax to describe instances of ER and REL in the subsequent. Figure 2 shows an instance of ER on the left-hand side and an instance of REL on the right-hand side. The graphical notation for ER is as usual. For both data models, key attributes are depicted underlined.

![Listing 2: Constraints of REL](image_url)

2.2 Transformation ER2REL

In Fig. 2, the REL model depicted is a valid refinement of the ER model. In fact, the REL model can be derived from the ER model. We can describe this transformation from ER to REL using ATL.

Listing 3 shows an example ER2REL of such an ATL transformation. The example uses some but not all of the ATL features. A complete description of the ATL language can be found in [20].
The ER2REL transformation definition is given in a module named ER2REL (first line) and transforms instances of the INput metamodel ER into instances of the OUTput metamodel REL (second line). The transformation consists of four rules S2S (schema to schema), E2R (entity to relation), R2R (relationship to relation), and A2A (attribute to attribute). ATL provides several kinds of rules, in our example we only use two of them: matched rules (S2S, E2R, and R2R) and lazy rules (A2A). These are the most commonly used kinds of ATL rules. Their semantics is as follows:

A matched rule is composed of a source pattern and a target pattern. The source pattern specifies a set of objects of the source metamodel and possibly guarding OCL expressions. The target pattern specifies a set of objects of the target metamodel plus a set of bindings. The bindings describe assignments to the features (attributes, references, association ends) of the target objects.

The semantics of matched rules is a two-step one: first, the source patterns of all guarded rules are matched against the input model. For every match, the objects of the target pattern are created. In the second step, the bindings for the target patterns are executed. In this step, a source-target resolution is applied: If the evaluation of a right-hand side expression of a binding evaluates to instances of one of the source metaclasses and if this instance has been mapped to an instance of a target metaclass by a matched rule in the first step, then the source instance value is replaced by the corresponding target instance value in the binding process. For example, rules E2R and R2R map ER entities and relationships to REL relations (please ignore their binding sections for now). Rule S2S maps ER database schemas to REL schemas. The binding section of this rule assigns the union of entities and relationships to the relations property of the REL schema. After evaluating the entities (\(\text{union(s.relations)}\)) the ER_Entity and ER_Relationship values in the result set are replaced by the corresponding REL_Entity and REL_Relation objects that have been created by rules E2R and R2R in step one.

Lazy rules such as rule A2A are treated differently. These rules are not applied automatically to all matches of their source patterns, they are applied explicitly. Rule A2A creates a REL attribute for an ER attribute (assigning the same name and isKey property to it). It is triggered in rules E2R and R2R using the thisModule\_A2A(a) invokes A2A for each ER attribute of the entity s. Considering keyER attributes (having isKey = true), two REL attributes are created for each of them using A2A, one by rule E2R and one by rule R2R (the latter ones are the foreign key attributes).

Further concepts of ATL that are not used in this example include OCL helper operations, called and unique lazy rules, and imperative sections in the rules.

So far we have introduced the metamodels ER and REL and the transformation ER2REL. Applied to our example (Fig. 2), the transformation ER2REL will generate the (valid) REL instance from the (valid) ER instance. The interesting question is: will it derive a valid REL instance for every valid ER instance? Put in more technical terms: Will the transformation result \(\sigma_{ER2REL}(\sigma_{ER})\) fulfill all constraints of REL for every instance \(\sigma_{ER}\) of ER that fulfills all constraints of ER?

In the next section we discuss how this and other questions can be addressed by translating the executable transformation (described by ATL rules) into an equivalent structural transformation model.

3. TRANSLATING ER, REL AND ER2REL INTO A TRANSFORMATION MODEL

The general idea is to combine the syntax of the source and target models and the execution semantics of the ATL transformation into an integrated structural model – the transformation model. The transformation model \(TM(ER, REL, ER2REL)\) contains all metaclasses and constraints of ER, as well as all metaclasses and constraints of REL. In addition, it contains synthesized metaclasses and constraints for the rules of ER2REL. Figure 3 shows the complete transformation model \(TM(ER, REL, ER2REL)\), the metaclasses for ER2REL are depicted using thick lines.

An instance of the transformation model represents the application of the transformation, materializing the trace between the source and the target. Coming back to our initial example from Fig. 2 we have one application of the matched rule S2S, two applications of rule E2R, one application of rule R2R and five applications of rule A2A to create the target model from the source model. Figure 4 shows this transformation as an instance of the transformation metamodel. This instance of \(TM(ER, REL, ER2REL)\) corresponds exactly to the application of ER2REL to the instance of ER depicted on the left-hand side, with the target model being on the right-hand side.

However, the transformation model described so far will allow several instances \(\sigma_{ER}\) that do not correspond to a tuple \(\sigma_{REL}\) such that \(\sigma_{REL} = ER2REL(\sigma_{ER})\). We require further constraints to make \(TM(ER, REL, ER2REL)\) equivalent to ER2REL. These constraints are shown in Listing 4. Summarized, the constraints should equivalently encode the semantics of the transformation ER2REL such that, given valid instances \(\sigma_{ER}\) and \(\sigma_{REL}\),

\[
\exists \sigma_{ER2REL} : \text{Valid}(TM(ER, REL, ER2REL))(\sigma_{ER}, \sigma_{REL}, \sigma_{ER2REL})
\]

Thus, if \(\sigma_{REL}\) is the result of \(\sigma_{ER}\) then a corresponding instance of the transformation model (containing \(\sigma_{ER}\) and \(\sigma_{REL}\)) exists and vice versa. In this sense, \(TM(ER, REL, ER2REL)\) is equivalent to ER2REL.

The individual constraints are discussed in the following. For the three matched rules (S2S, E2R, R2R) we have one constraint (_MATCH) each to ensure that the rule is applied on every applicable object exactly once. There is no such constraint for the lazy rule A2A, as this rule might be applied several times on the same source object. Consequently, we might have none or more than one A2A objects referring to the same ER_Attribute object.

We also need constraints (_CREATE) to ensure that every target object is created by exactly one of our four rules. Notice that REL_Relation objects can result from either ER_Entity or ER_Relationship, which is reflected in the '+' operation of invariant R_CREATE.

Finally, we have one constraint (_BIND) for each binding _-_ that assigns a value to one of the properties of the target objects. In general, all ATL bindings t.p <- expr become equality constraints t.p = expr. However, in the creation of the _BIND constraints, the resolution mechanism of ATL has to be encoded, as well as the explicit invocation of lazy rules. If the source expression is a regular OCL expression (not containing invocations of lazy rules) and the expression type is a primitive type or a collection or tuple type of a primitive type, this translation is straightforward, replacing _-_ by =. For example, the first binding name _-_ s.name in rule E2R of ER2REL becomes t.name = s.name in the invariant constraint E2REL_BIND_NAME.

ATL's implicit resolution for matched rules is replaced by explicit navigation from the source objects to the corresponding
module ER2REL;
create OUT : ER from IN : REL;

rule S2S { /* Schema to Schema */
from s : IN!ER_DBSchema
to t : OUT!REL_DBSchema ( relations <- s.entities->union(s.relships) )}

rule E2R { /* Entity to Relation */
from s : IN!ER_Entity
to t : OUT!REL_Relation ( name <- s.name,
attrs <- s.attrs->collect(a|thisModule.A2A(a)) )}

rule R2R { /* Relationship to Relation */
from s : IN!ER_Relship
to t : OUT!REL_Relation ( name <- s.name
atattrs <- s.attrs->collect(a | thisModule.A2A(a))->union(
s.ends->collect(re | re.entity.attrs)->flatten()->select(a|a.isKey)->collect(a|thisModule.A2A(a))) )}

lazy rule A2A { /* Attribute to Attribute */
from s : IN!ER_Attribute
to t : OUT!REL_Attribute ( name <- s.name, isKey <- s.isKey )}

Listing 3: ER2REL.atl

target objects. This is achieved by navigating over the meta-
classes that correspond to the applicable matched rules. For ex-
ample, in rule S2S, the OCL expression s.entities is replaced
by s.entities->collect(e| e.e2r->any(r| r.ctx = self.ctx).t) in the invariant S2S_BIND_NAME to
identify the target object that is created for respective source ob-
ject in the context of the current transformation. Notice that the
->any expression is deterministic here, because there is always ex-
actly one such E2R object in a valid instance of the transformation
model due to the E2R_MATCH constraint.

For the explicit application of lazy rules by thisModule, the
context of the application has to be considered when trans-
slating the binding into a constraint. This is required to
uniquely identify the corresponding rule application object. Con-
sider the bottom of Fig. 4: the ER attribute for taxId is
mapped onto two REL attributes by the lazy rule A2A. How-
ever, they were created in different contexts (rule E2R resp.
rule R2R). Therefore, a corresponding enumeration A2ACtx is
included in the transformation model to capture the creation
context for the lazy rule A2A. Consequently, the source ex-
pression s.attrs->collect(a | thisModule.A2A(a)) in rule
E2R has to be replaced by a.a2a->any(r| r.ctx = self.ctx
and r.lazyCtx = #E2R_1).t in the invariant constraint RB_I2,
while the same source expression occurring in rule R2R
has to be replaced by a.a2a->any(r|r.ctx = self.ctx and
r.lazyCtx = #R2R_1).t. The trailing number in the enumera-
tion literals indicate the position of the application within the re-
spective rule. In our example, the literal #R2R_1 identifies a map-
ning from attributes of a relationship to relation attributes, while
#R2R_2 identifies a mapping for the foreign key attributes of a re-

Figure 3: Transformation metamodel

Please notice that the translation scheme introduced in this paper
is not yet universally valid. In general, several other aspects have
to be considered, such as overlapping patterns of matched rules and
more generalized invocations of lazy rules, to name a few. In future
work, we are going to provide a complete translation scheme.

However, for the concrete example ER2REL we have given an
equivalent transformation model. In the next section we illustrate
how the transformation model can be used to validate the model
transformation.
4. USING THE TRANSFORMATION MODEL FOR VALIDATION

Due to its integrated structural nature, a transformation model is well-suited for analysis. As one example of how the transformation model can be used for the validation and analysis of a transformation, we will investigate the totality [10] of ER REL: Is there a valid model of ER that is transformed into an invalid model of REL by the transformation ER REL? In the sense of the transformation quality, the existence of such a model would be a strong counter argument against transformation correctness.

The question can be easily investigated using the transformation model: We have to look for an instance \( \sigma_{TM} \) of \( TM \) such that \( \sigma_{TM} \) conforms to the constraints of \( ER \) (ER EN, \ldots, ER K) and to the constraints that correspond to ER REL (S2S MATCH, dots, A2A BIND), but not to all constraints of REL (REL RN, REL AN, REL K).

Bounded verifiers for OCL-annoted UML models such as UML-toCSP [11], the USE generator [18], and UML to Alloy translations [1] can be used to automatically find such instances.

Notice that if we find a violating instance, no backwards translation of any kind is required to obtain the corresponding source and target models. We only have to omit all elements that do not belong to the source resp. target metamodel.

4.1 Detecting Identical Names for Relations

We investigate the three constraints REL RN, REL AN, and REL K of REL separately. To find a violation of the first constraint, we are looking for \( \sigma_{TM} \) such that

\[
ER_{EN}(\sigma_{TM}) \land \cdots \land ER_{K}(\sigma_{TM}) \land S2S\text{-MATCH}(\sigma_{TM}) \land \cdots \land A2A\text{-CREATE}(\sigma_{TM}) \land 
\neg REL_{RN}(\sigma_{TM})
\]

The negated version of REL RN is:

\[
REL_{DBSchema}.allInstances().exists(self | 
not self.relations->forAll(r1,r2 | r1.name = r2.name implies r1=r2))
\]

We use the search bounds depicted in Table 1 in the search for \( \sigma_{TM} \). Even in this small search space, instances of TM(ER, REL, ER REL) that are well-formed w.r.t. all constraints except REL RN can be found. Figure 6 shows one of them using the concrete syntax of ER and REL. We can see that the relationship and one entity share the same name (A) on the source side, which is allowed by ER. The transformation creates two relations sharing the same name, which is not allowed by REL, or more specifically, by REL RN. We can think of three options to fix this problem:

1. Disallow entities and relationships sharing names in ER (ie., fix the ER metamodel).
2. Qualify the generated relation names (ie., fix the ATL transformation ER REL).
3. Disallow entities and relationships sharing names in the transformation (ie., fix ER REL such that it raises an error if this condition is detected).

4.2 Detecting Identical Attribute Names

Secondly, we are checking for a violation of the constraint REL AN of REL. This time we have to find a state \( \sigma_{TM} \) that fulfills the following condition:

\[
ER_{EN}(\sigma_{TM}) \land \cdots \land ER_{K}(\sigma_{TM}) \land 
RA_{I1}(\sigma_{TM}) \land \cdots \land RC_{I2}(\sigma_{TM}) \land 
\neg REL_{AN}(\sigma_{TM}).
\]

The negated version of REL AN is:

\[
REL_{Relation}.allInstances().exists(self | 
not self.attrs->forAll(a1,a2 | a1.name = a2.name implies a1=a2))
\]

Table 1: Search bounds

<table>
<thead>
<tr>
<th>Modeling concept</th>
<th>Search bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of instances per class</td>
<td>0\ldots 5</td>
</tr>
<tr>
<td>Links per association</td>
<td>Any subset of the cartesian product of the corresponding class extents</td>
</tr>
<tr>
<td>Values for String-typed attributes</td>
<td>'A', \ldots, 'Z'</td>
</tr>
</tbody>
</table>
We can find such instances \( \sigma_{tm} \) using the same search bounds as for REL_RN. Figure 7 shows one example. For relationship C, both participating entities have the same key attribute names. Therefore, the corresponding relation C has two attributes with the same names, which is not valid in REL. This is (obviously) a bug in ER2REL. It has to be fixed, for example, by qualifying the foreign key attributes by the role name (in this case: \( V.U, W.U \)).

4.3 Detecting Missing Keys

Finally, a state \( \sigma_{tm} \) violating (only) invariant REL_K can be found in the search bounds, too. Figure 8 shows an example. In this case, the problem is that one of the entities does not have a key attribute. Using our simple version of ER2REL, this results in an inadequate translation for B and C. This is inadequate for B because two distinct instances of the entity type B sharing the same attribute values could not be distinguished in relation B. Furthermore, B is not referenced in relation C. A reasonable fix for ER2REL would be to generate an artificial identifier attribute in the relation for every entity without key attributes.

4.4 Further Validations / Verifications

In the preceding we stressed the validation of the totality of ER2REL. However, this is only a first example, several other questions regarding ER2REL can be addressed using TM(REL,ER,ER2REL) in a similar way.

Further formal properties of the transformation can be validated (or verified) by adding further OCL constraints to the transformation model. [10] provides an extensive list of such properties, for example, if the transformation is exhaustive, or bijective.

We also see several useful applications of transformation models in interactive environments, guiding the transformation developer. For example, even if the original transformation is one-way, we can use the transformation model (in conjunction with a bounded model consistency checker) to search for source models that are transformed into a provided target model. This can be useful in reverse engineering scenarios: Given a hand-crafted relational database schema, from which ER models could it be derived?

An important aspect is that transformation models provide a uni-
ties such as confluence and termination for graph grammar based model transformation, [14, 3, 21, 2] who prove formal proper-
of M2M transformations include the works of [28] and [7] who any underlying transformation in another formalism.

As far as we know, our approach is the first one that systematically integrates the precise syntax of the source and target metamodels and the execution semantics of ATL in an transformation model which is equivalent to the underlying transformation. In the context of target model consistency, our approach addresses the systematic barrier of test case generation as described in [5].

5. RELATED WORK

There are several kinds of contributions that can be related to our work. Our work closely relates to [10] and [19]. Both approaches express model transformations by transformation models, employing OCL constraints.

The proposal in [10] derives transformation models from QVT Relations and triple graph grammars in a similar way as we do for ATL transformations to verify several formal transformation properties. Our proposal can be seen as an extension of this approach to ATL. [19] shows how to use a transformation model in the testing of ATL model transformations. However, while our transformation model is an equivalent (derived) representation of the underlying model transformation, the transformation model in [19] is used to express specific transformation contracts, similar to [13, 4, 12]. In this sense, our proposal can be seen as a white-box approach to validation and verification, whereas transformation contracts are a black-box approach.

The work in [17] discusses the various relationships between the ER and relational syntax and semantics as one integrated transformation model which, however, is hand-crafted and not derived from any underlying transformation in another formalism.

Further white-box approaches to the validation and verification of M2M transformations include the works of [28] and [7] who translate ATL rules into rewriting logic to analyze and validate model transformation, [14, 3, 21, 2] who prove formal properties such as confluence and termination for graph grammar based model transformations, and [23] who introduce a transformation model checker to build a state space for the transformation language DSLTrans.

Further black-box approaches comprise the generation of suitable test cases [27, 8, 15] and the use of transformation pre- and postconditions for the validation of model transformations [24, 16].

As far as we know, our approach is the first one that systematically integrates the precise syntax of the source and target metamodels and the execution semantics of ATL in an transformation model which is equivalent to the underlying transformation. In the context of target model consistency, our approach addresses the systematic barrier of test case generation as described in [5].

6. CONCLUSION AND FUTURE WORK

This paper discussed two questions. First, we explained how ATL transformation rules can be translated into an equivalent transformation model. Thus we translated the executable description into an equivalent structural one. The result is an integrated model containing the precise syntax of the source and target metamodels as well as the precise semantics of the model transformation.

Second, we have shown how the transformation model can be used to analyze the original transformation. We discussed how model consistency checking tools can be employed to investigate the transformation model. For our example ER2REL, we validated whether the transformation is total by systematically checking for violations of the target metamodel constraints. We could find three shortcomings of the ATL transformation. As for all model checking approaches, the search bounds limit our approach: If we can find a valid source model that is transformed into an invalid target model, we know that there is a flaw in the ATL rules considered. If we cannot find such a source model it only means that no such model exists within the search bounds. It may exist outside the search bounds. Tools for the formal verification of OCL-annotated UML models such as [9, 22, 26] could be employed to overcome this restriction. In general these approaches still require interaction with the modeler and a strong understanding of the underlying proof formalism (e.g., Isabelle HOL), although [26] promises automated reasoning for a rich subset of OCL.

We want to stress that there is complementary relationship between the executable model transformation and the corresponding transformation model. While the executable definition (in this case: in ATL) provides an effective and usually efficient way to describe the transformation, it is not accessible for analysis. In contrast, transformation models are not directly executable. Given a source model, the corresponding target model could be found by a model consistency checker, but this certainly is not an efficient approach. But transformation models provide a representation that is very accessible for analysis, as illustrated in this paper. Furthermore, the interpretation of a transformation model is very intuitive as it materializes the trace between the source and target model elements.

Our next research objectives are: (1) Provide a systematic translation of ATL transformations into transformation models that is as complete as possible, but excluding the imperative extensions of ATL. (2) Elaborate how further transformation properties can be analyzed using transformation models. In particular, we are interested in multi-formalisms settings, where we consider transformation models to be a unifying approach.

7. REFERENCES

