A metrics-driven inspection framework for model transformations

Granda, Maria Fernanda; Parra, Otto; Condori-Fernández, Nelly

2019

document version
Publisher's PDF, also known as Version of record

A Metrics-driven Inspection Framework for Model Transformations

Conference Paper · April 2019

CITATIONS
0

READS
112

3 authors:

María Fernanda Granda
University of Cuenca
15 PUBLICATIONS **71** CITATIONS
SEE PROFILE

Otto Parra
University of Cuenca
13 PUBLICATIONS **11** CITATIONS
SEE PROFILE

Nelly Condori-Fernández
University of A Coruña
101 PUBLICATIONS **846** CITATIONS
SEE PROFILE

M. F. Granda, O. Parra, N. Condori-Fernandez. A Metrics-driven Inspection framework for Model Transformations. Proceedings in XXII Ibero-American Conference on Software Engineering. La Habana-Cuba. 22-26 April, 2019.

A Metrics-driven Inspection Framework for Model Transformations

Maria Fernanda Granda[1], Otto Parra[1], and Nelly Condori-Fernández[2,3]

[1] Department of Computer Science, University of Cuenca, Cuenca, Ecuador
{fernanda.granda, otto.parra}@ucuenca.edu.ec
[2] Department of Computer Science, University of A Coruña, Spain
n.condori.fernandez@udc.es
[3] Vrije Universiteit Amsterdam, The Netherlands
n.condori-fernandez@vu.nl

Abstract. [Context] Model transformations are key elements of Model-driven Engineering. They allow querying, synthesizing and transforming models into other models or code. [Problem] However, as with other software development artefacts, they are not free from anomalies and thus require both verification and validation techniques. [Objective] The objective of this study is to define a semi-automated framework for inspecting the correctness (notions of type and correspondence) of model transformations, by means of detecting and locating anomalies in the transformation rules. [Method] In order to compare the correctness of source and target models, we assume that operational behaviour can be compared by metrics applied on projections from the source model to the target (with deliberate loss of information), which should be preserved by the transformation. [Results] We demonstrate the applicability of our framework for inspecting the correctness of a model-to-model transformation required in a model-driven testing approach. The main result of the study highlights the advantages of metrics for detecting any missing, incorrect or unnecessary transformation rules that have an impact on the correctness of the model transformations. From the research perspective, the feedback produced by the implemented tool will be useful for future research.

Keywords: Model Transformations, Type-Correctness, Correspondence Correctness, Inspection, Metrics.

1 Introduction

Models are the main artefacts in Model-Driven Engineering (MDE) and efforts are focused on their creation, testing and evolution at different levels of abstraction through model transformations. However, as with other software development artefacts, model transformations are not anomalies-free and so must be systematically verified and validated [1]. Otherwise, anomalies introduced by the transformations will be propagated and may produce more anomalies in the subsequent MDE activities.

Verification and validation are independent procedures that are used together for checking that a product (i.e. software artefact) meets requirements and specifications and that it fulfills its intended purpose. However, verifying a transformation that transforms one artefact into another is fundamentally more complex than verifying an individual artefact itself [2]. Thus, some researchers argue that specialist techniques are required [1][2][3] to this task. The inspection is used in researches to validate a software artefact, which is analysed against predetermined criteria of entry or against the specifications which were used to build the artefact. Applying inspection technique to the model transformation validation, one has the documentation of the target model as predetermined criterion, which will be compared with the produced model. These models (i.e. source and target model) are considered the entries, so that the comparisons are done. The goal of our research is to make both *type-correctness* property and *correspondence correctness* of model transformations measurable.

But, what is a "correct" transformation? We considered some of the different notions of correctness summarized by Rahim and Whittle [1]. These authors claim that a transformation satisfies the *type-correctness* property if its target models is conforming to the abstract syntax of the target language. A transformation is said to be correct with respect to *static semantics* if the target models satisfy the well-formedness constraints of the target metamodel. A transformation is correct with respect to *dynamic semantics* if the target models preserve a given property of the source model (these could be domain properties of the source model such as security, application-specific properties, or properties relating to the semantics of the source modelling language, e.g., run-to-completion semantics for UML state machines). A transformation can also be deemed correct if the target model contains the expected target elements corresponding to the source elements in the source model. They refer to this aspect of correctness as *correspondence correctness*. They also acknowledge that some approaches focus on semantics of model transformation properties of the transformation itself—such as termination, confluence of transformation rules, and executability.

In this paper, we present a semi-automated metrics-driven inspection framework to measure both *type-correctness* property and *correspondence correctness* of a complex model transformation used in the context of Model-Driven Engineering. With the purpose of demonstrating the applicability of our framework, it was used in an model-driven testing approach named CoSTest [4][5]. The main contributions of this paper are:

- A semi-automated process defined for inspecting the correctness of model transformations.
- Automated tool to support some activities of such process (i.e. model transformation and report generation).
- A set of metrics defined to predict the quality properties we propose to measure (i.e. *type-correctness* property and *correspondence correctness*). These metrics are evaluated at the instance level, meaning that each execution of the transformations is measured.

This paper is structured as follows: Section 2 summarizes the work related. Section 3 presents the framework and the set of proposed metrics to measure the model transformation correctness. In section 4, we demonstrate the applicability of our framework using an illustrative example. Finally, Section 5 outlines the conclusions and future work.

2 Related Work

Research in validation using metrics and inspection is still at an early stage, where the research focuses on establishing metrics and mapping them to relevant quality attributes of model transformations such as maintainability, testability, performance. A survey [1] summarizes 57 approaches for verifying model transformations and classify them along two dimensions. Firstly, the authors present a coarse-grained classification based on the technical details of the approach (i.e., testing, theorem proving, graph theory, model checking, inspection and metrics). Secondly, they present a finer-grained classification which categorizes approaches according to criteria such as technique, effort, tooling, properties verified and type of transformation.

From these results, we can summarize that only eight approaches are related to inspection and metrics of model transformations (see Table 1). In most of them (7 out of 8 approaches), the technique is applied on the transformation rules (i.e. direct technique) instead of verifying the properties of the generated output (i.e. indirect technique). Column 3 in Table 1 shows that only 4 approaches have a tool support. Some works (3 out of 8 approaches, see column Property in Table 1) are focussed in the preservation of static semantics of models, one approach does not report the property and others (4 of 7 approaches) addressed the preservation of dynamic semantics of models. About the types of transformation for which the related work is applicable, the survey reported 5 of 8 approaches addressing model-to-model (M2M) transformation, 1 approach for model-to-code (M2C) transformation and 2 approaches for both type of transformations (see fifth column in Table 1).

Table 1. Categorization of inspection approaches taken from [1]

Approach	Technique	Tool	Property	Transformation Type	Metrics-driven
[6]	Indirect	No	-	M2C	No
[7]	Direct	Yes	Termination, confluence, executability	M2M	Yes
[8]	Direct	Yes	Termination, confluence, executability	Both	No
[9]	Direct	Yes	Termination, confluence, executability	M2M	No
[10]	Direct	No	Dynamic semantics of models	Both	Yes
[11]	Direct	No	Dynamic semantics of models	M2M	Yes
[12]	Direct	Partial	Dynamic semantics of models	M2M	Yes
[13]	Direct	No	Dynamic semantics of models	M2M	Yes
Our proposal	Indirect	Partial	Correctness of type and correspondence	M2M	Yes

Currently indirect approaches using theorem proving are focused on verifying M2M transformations, and these approaches require the translation of the target model into a

formalism (e.g. semantic model) that can be handled by a theorem prover [1]. Theorem provers are either used to prove that the model transformations can generate target models with certain properties or compare the target models with oracles [1].

In contrast to this type of indirect techniques that use theorem proving, we proposed a semi-automated framework for verifying and inspecting the correctness of type and correspondence for M2M transformations, which consists of (i) a tool to generate a report and support the manual intervention to inspect the target model comparing with the expected model (oracle); (ii) set of proposed metrics providing a measure of the model transformation correctness achieved. These values then will be used to locate the anomalies in transformation rules and correct them. Next section we present the details of our approach.

3 Framework for inspecting the Correctness of Type and Correspondence of Model Transformations

A conceptual overview for a model-to-model transformation (M2M) from a modelling language A to B is given in Fig. 1. Two important elements must be stuck out in this context: (1) ***Metamodel*** constitutes the definition of a modelling language, which provides the constructor and the relations with constraints for describing the whole class of models that can be represented by that language with a valid semantic. (2) A ***transformation*** defines a correspondence relation (e.g. transformation rules) between elements in a source metamodel A and elements in a target metamodel B. Therefore, executing transformation A2B helps build a group of elements in target models to conform (well-formed) to their metamodels, using the information from a set of elements in source models, which must also agree with their metamodels.

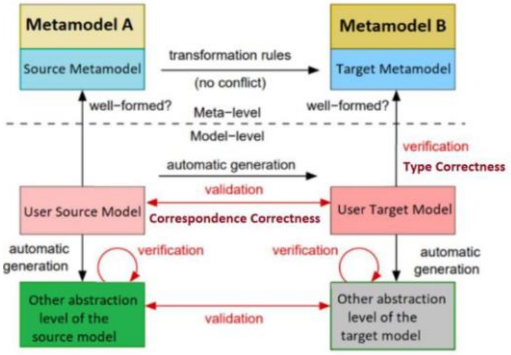

Fig. 1. Model transformation adapted from [14]

Our inspection framework focuses on two aspects: type-correctness and correspondence correctness for model transformations. For the *type-correctness*, we verify whether a model transformation can generate well-formed target models. Well-formedness is verified by checking whether the target model conforms to OCL constraints of the target metamodel. For the *correspondence correctness*, we compare (i.e. inspect) the structural correspondence between the expected (i.e. oracle) and target

models. Therefore, we applied an indirect technique. Fig. 2 shows an overview on how our framework operates, its (manual and automatic) activities, input and output artefacts. A description of each activity of our proposed framework is as follows:

1. Defining precisely each modelling languages (both A and B) based on metamodeling. It is required include traceability information in target metamodel (see Section 4.1 for an example).
2. Defining the model transformation using a specific set of structural correspondence rules (i.e. non-conflicting mappings or transformation rules) to transform the source model into target model (A2B transformation).
3. Deriving and verifying the target model. For any specific (but arbitrary) well-formed model instance of the source language A, we derive and check if the corresponding B target model satisfies the well-formedness constraints of the target metamodel (static semantics) by automatic transformation (e.g., using ATL [15]).
4. Traversing the models and collecting the cross-links (i.e. model attributes) to trace source elements with the corresponding target elements.
5. Measuring the correctness (type and correspondence) of the model transformation by applying inspection and the proposed metrics.
6. Taking a decision, although the approach can indicate the correctness level of the model transformation, the inspector (expert) is who takes a decision for considering if the target model is correct. If so, then indirectly the transformation and its rules can be considered are correct as well.
7. In another case, the transformation rules have to be adjusted and the evaluation process has to be executed again. For this purpose, the report generated in the step 4 helps to locate the anomalies in the transformation rules.

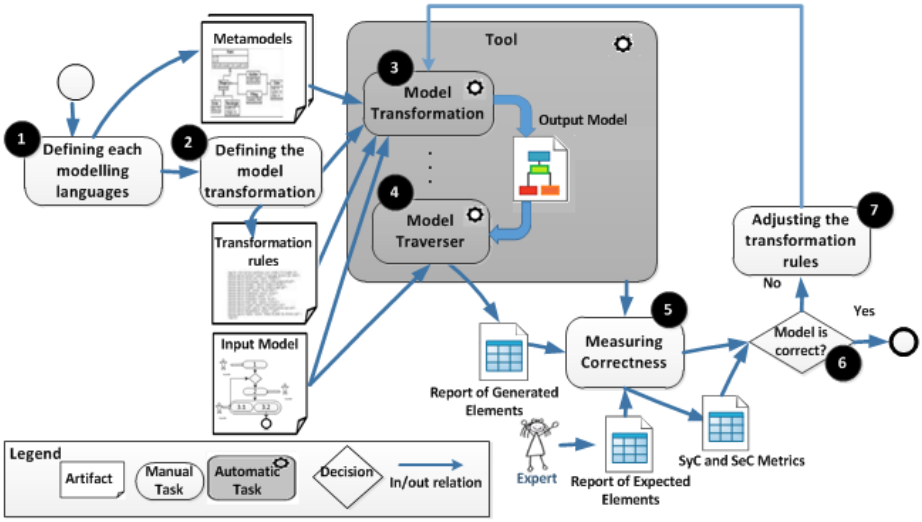

Fig. 2. Overview of our framework for inspecting the correctness of a M2M transformation

3.1 Automated tool for inspecting the target model and traversing the models

The M2M transformation with its transformation rules can be evaluated by performing a simple depth-first search on the model instances. Our tool gives support to both third and fourth activities of our framework (See Fig. 2), by automating the model transformation and generating reports with the trace of the relationship between the source and target elements of the models. The model transformations were implemented using an ATL (ATLAS Transformation Language) [15] and metamodels in Meta Object Facility –MOF [16], the Eclipse Modelling Framework -EMF [17] that employs constraints using Object Constraint Language -OCL [18] to precisely describe their invariants.

3.2 Metrics for evaluating the model transformation correctness

As shown in Fig. 2 the report of generated elements (outcome of the tool) are input of the correctness measurement activity. In this sub-section, we present the metrics with scope of both transformation and rule proposed to measure the type correctness and correspondence correctness of model transformations.

Definition of Basic and Derived Metrics with Transformation scope.

The ***type-correctness*** of a transformation (TC_T) is measured by the metric value achieved by the respective rules in the target model. *ATrule* measures the correctness of an atomic rule (it does not contain any reference to any rule including self-reference, while *Crule* measures the correctness of the composite rule (*CompositeRule*), because its result depends on two values (1) outcome of its nested rules, which generates elements on target model that need others nested rules; and (2) the own *ATrule* value.

If an atomic rule generates a correct element, then its *ATRule* value is 1; otherwise its value is 0. Since target model can have a hierarchical structure, TC_T is calculated starting from the most nested level of the structure up to the highest level. These derived metrics are as follows:

- **Type-Correctness for rule *i***, the Crule*i* value is calculating using the average of the sum of its ATrule*i* values plus the average of its k element values corresponding to Crule*l* or ATrule*l* of the nested rules.

$$Crulei = \left(\text{ATrule}i + \frac{\sum_{l=1}^{k} \text{Crule}l \mid \text{ATrule}l}{k}\right)/2 \qquad (1)$$

- **Type-Correctness of the model Transformation** (TC_T) corresponds to the ACrule value in the top level of the model. This value is multiplied by 100 to obtain the percentage of the type-correctness of the transformation.

$$\text{TC\_T} = \text{Crule}i * 100\% \qquad (2)$$

For ***Correspondence Correctness*** (CC_T) of a transformation, a similar pattern to that for metrics on Type-Correctness is followed. However, we only consider one value for correspondence correctness of a *CCrule*, if the rule is a *CompositeRule*, we take the

composite *CCrule* value; otherwise we take the *ATRule* value. The metrics that can be used in both transformations are as follows:

- **Correspondence Correctness for rule *i***, the CCrule*i* value is calculated using the average of the sum of its k element values corresponding to CCrule*l* or ATrule*l* of the nested rules.

$$CCrulei = \frac{\sum_{l=1}^{k}[CCrule l \mid ATrule l]}{k} \quad (3)$$

- **Correspondence Correctness of the model Transformation** (CC_T) corresponds to CCrule value in the top level of the model. Both values are multiplied by 100 to obtain the percentage of the correspondence correctness of the transformation.

$$CC\_T = CCrulei * 100\% \quad (4)$$

Definition of Basic and Derived Metrics with Rule scope.

The basic metrics with rule scope are shown in Table 2. Expected elements are provided by the expert. On the other hand, the elements generated by each rule are retrieved from the target model using the information of traceability (i.e. rule used) stored in each model element. This report is supported by our tool. The respective derived metrics (ratio values) are listed in Table 3. The values of these metrics are only at the element level, so that they do not consider the contained elements. The contained elements are considered in the next metrics with transformation scope.

Table 2. Basic metrics to calculate the Correctness of Type and Correspondence of a Rule

Metric	Definition
N_EG*j*	Total number of elements generated by the rule j
N_CEG*j*	Total number of Correct Elements generated by the rule j
N_EE*j*	Number of expected elements to be generated by the rule j

Table 3. Derived metrics to calculate the Correctness of Type and Correspondence of a Rule

Metric	Definition	Formula	
TC_*rulej*	Type-Correctness reached by the rule j	N_CEG*j* / N_EG*j*	(5)
CC_*rulej*	Correspondence Correctness reached by the rule j	N_EG*j* / N_EE*j*	(6)

Thanks to the measures (outcome of the correctness measurement activity), the inspector (expert) has the information to realize on the differences between the obtained model and the expected one. These differences could mean: a) there are unnecessary rules in the M2M transformation that generate additional elements of the expected ones, b) there are missing rules because the elements were not generated as expected, or c) the rule is incorrect, the expected output may be different from the actual output because an existing rule is not correctly implemented. However, these measures do not to allow knowing if the errors are serious or insignificant. In this paper, we considered all errors with the same severity, but we could define a weight for each rule type in order to classify the error relevance.

In the following section, we illustrate each activity of our metrics-driven inspection framework in the context of model-driven testing, where test case models are automatically derived from requirements models.

4 Application of the inspection framework

We present (an extract of) a complex model transformation from Communicational Analysis (CA) Model [19] to CoSTest's Test Model (TM) [4] (denoted as CA2TM) in order to demonstrate the applicability of our framework for measuring the correctness of model transformations. We selected this transformation (CA2TM) because during the development phase of the CoSTest tool, we were challenged to measure the correctness of the model transformations implemented in the tool. Therefore, we collected the process and data followed for inspecting and measuring the M2M transformation, which are reported in this paper.

4.1 Defining modelling languages

CA Model as the source modelling language. CA is used to specify the requirement models and includes a Communicative Event (CE) and the message structures as main artefacts for specifying the functional requirements. Fig. 3 shows the main CA artefacts based on the Sudoku system [20], which defines the functionality for managing different users, playing with their sudokus and generating new ones.

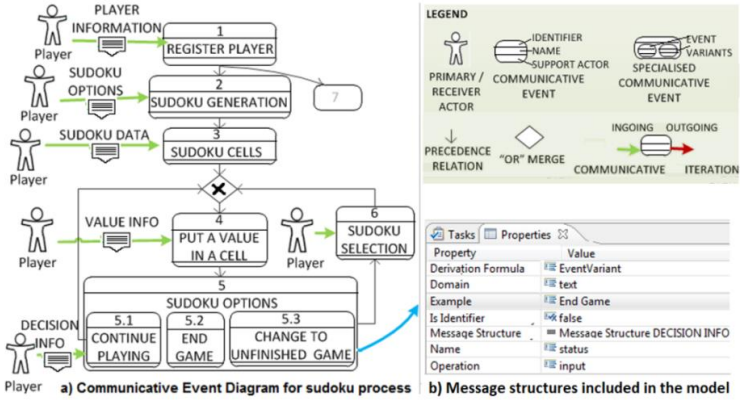

Fig. 3. Excerpt from a CA model for Sudoku CS, adapted from [20]

CoSTest's TM as the target modelling language. CoSTest's TM is used to capture the semantics of the test model available in the CoSTest tool. A precise metamodeling treatment of CoSTest's TM was discussed in [4]. However, we have updated this metamodel including in Element class the traceability information of each one of them (i.e. location and transformation rule attributes) in order to measure the proposed correctness metrics. Now, we briefly revisit the metamodel and the operational semantics of CoSTest's TM in Fig. 4, it includes the rules that generate the different target model elements.

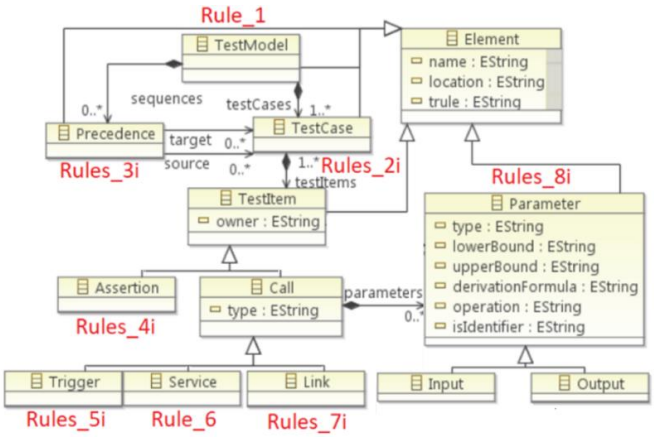

Fig. 4. CoSTest metamodel for our transformation example (taken from [4])

4.2 Defining the CA2TM model Transformation

For the sake of readability, we use concrete syntax to describe instances of requirements model (see Fig. 5a) and test model (see Fig. 5b) with two test scenarios (i.e. TS1 and TS2) for our example, Sudoku system. As in the CA to TM, model transformation can be used to generate a target model of a certain structure (TM) from a source model of a different structure CA. Specific structural configurations in the source model (such as an Communicative Event node in the CA model, (see Fig. 5) produce specific structural configurations in the target model (such as a Test Case in the TM).

The rules to accomplish the structural transformations may be simple (i.e. the mapping correspondence is 1 to 1) or complicated (i.e. the mapping correspondence is 1 to n) (see some examples in Table 4).

In essence, we expect that the correspondence conditions are independently specified for a model transformation, and an independent tool checks if these conditions are satisfied by the instance models, after the model transformation has been executed. We also require that the model transformation builds up a structure for bookkeeping the mapping between the source and target models (e.g. using cross-link attributes as the Element class shown in Fig. 4).

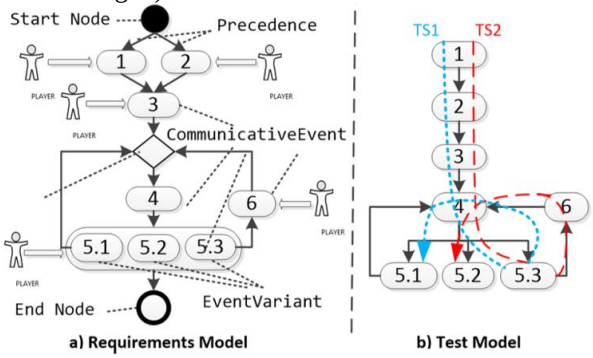

Fig. 5. Examples using graphical concrete syntax of (a) CA model, (b) CoSTest's TM

Table 4. Some transformations and mapping correspondences used in our CA2TM case study

CA	TM	mapping correspondence
Model	Test Model	1:1
Precedence	Precedence	1:n
Communicative Event	Test Case	1:1
	Assertion	1:1
Textual Requirement	Assertion	1:1
Iteration	Link	1:n
Node (End, Start)	-	Informational
Logical node (And, Or)	-	Informational
Organisational actor	-	Not used

4.3 Deriving and Verifying the Target Model

The transformation rules previously defined are formalized using ATL language, an XMI (XML Metadata Interchange) [16] representation then is generated automatically, which would yield the target model (Fig. 5b) as the output when supplying (the XMI representation of) the Sudoku's CA model (Fig. 3) as the input.

The target model can be checked automatically to prove any model property using existing model checker tools or invariants in the same transformation tool (see an example of invariants in Fig. 6). Fig. 6 shows some OCL invariants used as the static semantics that are checking when the target model (TM) in our CA2TM transformation is derived. These example of constraints include that names must be unique within their respective contexts, classes must have a name and the multiplicity constraints for relations, and so on.

```
context TestModel inv TC_mult1: self.testCases->size()>0;
context Element inv Element_name: self.name<>null;
context Element inv Element_location: self.location<>null;
context Element inv Element_trule: self.trule<>null;
context Precedence inv Precedence_Name: self->forAll(e1,e2 | e1.name = e2.name implies e1=e2);
context Precedence inv Preced_TC: self.target<>null or self.source<>null;
context TestCase inv TestCase_Name: self->forAll(e1,e2 | e1.name = e2.name implies e1=e2);
context TestCase inv TI_mult1: self.testItems->size()>0;
context TestItem inv TestItem_owner: self-> forAll(e|e.oclIsKindOf(Link) = false implies e.owner<>null);
```

Fig. 6. Some OCL invariants used for the target model of the CA2TM transformation

If the verification succeeds, then we conclude that the model transformation is correct with respect to static semantic of the pair (p, q) of properties for the specific pairs of source and target models having semantics defined by a set of transformation rules. Otherwise, property p is not preserved by the model transformation and debugging can be initiated based upon the error trace(s) retrieved by the model checker. As before, this debugging phase may fix problems in the model transformation or in the specification of the target language. However, there are anomalies on transformation rules that are difficult to detect such as elements that have type-correctness problems (e.g. element names are syntactically incorrect) as well as correspondence correctness problems (e.g. missing or unnecessary elements). Therefore, in the Section 3.2, we present a set of metrics that will help to detect this kind of problems.

4.4 Traversing the Models

Since the metamodels of both the source and target models are available with the transformations, and the trace information (i.e. *name*, *location* and *trule* in the *Element* class, see Fig. 4) is included in the target metamodel, we have implemented the code of a model traverser in our tool to trace source elements with the corresponding target elements, and generate a report for inspection. This trace information needs to be analysed each time the rules specification changes. We call the model traverser at the end of each execution of the transformation, supplying to it the source and target model instances with the trace information (see example presented in Section 4.1).

4.5 Measuring the Correctness of the CA2TM model Transformation

To show how metrics measure the correctness in a model, we will use the data of a partial test model TM1 of our illustrative example (i.e. Sudoku system), with four test cases (i.e. TC1 – TC4) and one precedence Pr1.

Firstly, we used the report of the elements obtained by the M2M transformation using the tool (see Tables 5 and 6). Second, the expert compares that the report of elements is equal to the expected report otherwise it has to be completed with the missing elements before calculating the metrics (e.g. row highlighted in Table 5 is added because it is a missing element in the output model). Third, the expert checks the output model with expected model and assign the values ATrulei for each model element. In our example, there are some elements that have type-correctness problems (e.g. parameter and trigger names are syntactically incorrect because they include spaces) as well as correspondence correctness problems (e.g. unnecessary elements). This can be clearly evaluated by assigning to these elements the value 0 to ATrulei, such as some values in Table 5 and Table 6 are shown.

Table 5. Report generated with our tool to calculate the metric TC_T for our illustrative example

TEST MODEL	Rule	Crulei (1)	ATrulei	Crule/	Element (Precedence or Test Case)	Rule	Crulei (1)	ATrulei	Crule/	Test Items (Assertion, Service, Trigger, Link)	Rule	Crulei (1)	ATrulei	Crule/	Parameters	Rule	Crulei (1)	ATrulei
TM1	R1	0,97	1	0,94	Test case TC1	R2.1	1,00	1	1,00	Service S1	R6.1	1,00	1	1,00	parameter P1	R8.1	1	1
					Test case TC2	R2.1	1,00	1	1,00	Service S2	R6.1	1,00	1	1,00	parameter P1	R8.1	1	1
					Test case TC3	R2.1	0,78	1	0,56	Trigger T1	R5.2	0,67	1	0,33	parameter P1	R8.1	1	1
															parameter P2	R8.1	0	0
															parameter P3	R8.1	0	0
										Trigger T2	R5.2	0,50	1	0,00	parameter P1	R8.3	0	0
										Trigger T3	R5.2	0,50	1	0,00	parameter P1	R8.3	0	0
															parameter P2	R8.3	0	0
					Test case TC4	R2.1	0,92	1	0,83	**Trigger T1**	**R5.2**	**0**	**0**	**0,00**	**parameter P1**	**R8.3**	**0**	**0**
										Link L1	**R7.2**	**1,00**	**1**	1,00	parameter P1	R7.2	1	1
															parameter P2	R7.2	1	1
										Link L2	**R7.1**	**1,00**	**1**	1,00	parameter P1	R7.1	1	1
															parameter P2	R7.1	1	1
										Service S1	**R6.1**	**1,00**	**1**	1,00	parameter P1	R8.1	1	1
															parameter P2	R8.1	1	1
															parameter P3	R8.1	1	1
															parameter P4	R8.1	1	1
										Assertion A1	**R4.4**	**1,00**	**1**					
										Assertion A2	**R4.4**	**1,00**	**1**					
					Precedence Pr1	**R3.1**	**1,00**	**1**										

TC_T= Crule1*100
TC_T= 97% (2)

From these results, Table 5 shows that the Type-Correctness of the transformation (TC_T) is 0.97=97%, we applied the metrics (1) and (2). In similar way, Table 6 shows

that the Correspondence Correctness of the transformation (CC_T) is 0.92=92%, we applied the metrics (3) and (4).

Then, we calculate the TC_rulej and CC_rulej metric values for each transformation rule by comparing the number of elements generated with the number of expected elements. Table 7 shows the results of calculating the metrics (5) and (6) for our transformation example using the data from Table 5 and Table 6.

From these results, we see that the rules 5.2 and 8.3 are incorrect (see values of TC_rulej column in Table 6) because they are generating elements with type-correctness anomalies (e.g. names with spaces). In the similar way, we see that the rules 6.2 and 7.2 are generating missing or unnecessary elements (see values of CC_rulej column in Table 6 and the anomalies identified in Table 5); therefore, these metrics are less than 100%. Since the rule 8.1 being nested in rule 6.2, which generates a missing element (see row of the missing trigger in Table 6), so its value also does not reach 100%.

Table 6. Report generated using our tool to calculate the metric CC_T for our illustrative example

TEST MODEL	Rule	CCrulei (3)	Element (Precedence or Test Case)	Rule	CCrulei (3)	Test Items (Assertion, Service, Trigger, Link)	Rule	CCrulei (3)	Parameters	Rule	CCrulei (3)	Atrulei	Anomaly
TM1	R1	0,92	Test case TC1	R2.1	1,00	Service S1	R6.1	1,00	parameter P1	R8.1	1	1	
CC_T= CCrule1*100			Test case TC2	R2.1	1,00	Service S2	R6.1	1,00	parameter P1	R8.1	1	1	
CC_T= 92% (4)			Test case TC3	R2.1	0,75	Trigger T1	R5.2	1,00	parameter P1	R8.1	1	1	
									parameter P2	R8.1	1	1	
									parameter P3	R8.1	1	1	
						Trigger T2	R5.2	1,00	parameter P1	R8.3	1	1	
						Trigger T3	R5.2	1,00	parameter P1	R8.3	1	1	
									parameter P2	R8.3	1	1	
						Service S1	**R6.2**	**0,00**	parameter P1	R8.1	0	0	Missing Trigger
			Test case TC4	R2.1	0,83	**Trigger T1**	**R5.2**	**1,00**	parameter P1	R8.3	1	1	
						Link L1	**R7.2**	**0,00**	**parameter P1**	**R7.2**	**0**	**0**	Unnecessary Link
									parameter P2	**R7.2**	**0**	**0**	
						Link L2	**R7.1**	**1,00**	parameter P1	R7.1	1	1	
									parameter P2	R7.1	1	1	
						Service S1	**R6.1**	**1,00**	parameter P1	R8.1	1	1	
									parameter P2	R8.1	1	1	
									parameter P3	R8.1	1	1	
									parameter P4	R8.1	1	1	
						Assertion A1	R4.4	1,00				1	
						Assertion A2	R4.4	1,00				1	
			Precedence Pr1	R3.1	1,00							1	

Table 7. Values of TC_rule*j* and CC_rule*j* calculated for our example

CA2TM Transformation rule	TC_rulej (1)	CC_rulej (2)	CA2TM Transformation rule	TC_rulej (1)	CC_rulej (2)
1	1/1= 100%	1/1=100%	6.2	-	**0/1= 0%**
2.1	4/4 = 100%	4/4 = 100%	7.1	3/3=100%	3/3=100%
3.1	1/1 = 100%	1/1 = 100%	7.2	3/3=100%	**0/3=0%**
4.4	2/2= 100%	2/2 =100%	8.1	**9/10=90%**	**9/10=0.90%**
5.2	**3/4=0.75%**	4/4=100%	8.3	**0/4=0%**	4/4=100%
6.1	3/3=100%	3/3=100%			

4.6 Taking a decision

The differences found allowed us to take corrective actions to adjust our M2M transformation, so that for the next iteration the problems identified in the

transformation rules were fixed, achieving 100% in the correctness of type and correspondence.

4.7 Adjusting the transformation rules

By using the report generated by our tool, we located easily the anomalies in the transformation rules, which facilitated the correction process.

Our framework was applied to CA2TM transformations with ten CA source models. Each CA source model contained a variety of characteristics that allowed us to test the functioning of the different transformation rules defined for CA2TM transformation under review. Due to severe page limitations, we can only provide an overview of a illustrative example of the CA2TM transformation, the reader is referred to URL (https://costestproject2017.wordpress.com/) for a more detailed information of the other CA2TM transformations. Then, the differences found in each iteration allowed us to take corrective actions to adjust the M2M transformation, so that for the next iteration the problems identified in the transformation rules were fixed. Achieving a correctness (type and correspondence) of 100% from the 5th CA2TM transformation.

5 Conclusions and Future Work

In this paper, we proposed a semi-automated metrics-driven framework to inspect if a model transformation preserves the type correctness and correspondence correctness. For this purpose, we have defined a set of basic and derived metrics, which can be computed and used to inspect the target model comparing with the expected model (oracle). The metrics-driven inspection is supported with a report implemented in the tool containing the elements of target and source models.

We demonstrated the applicability of our framework for inspecting the correctness of a complex model transformation from a requirements model (i.e. Communicational Analysis model) to test models (i.e. CoSTest's test model). We found that the defined metrics were useful for identifying and locating anomalies (i.e. incorrect, missing and redundant rules) in the transformation rules, which improved the effectiveness of the model transformation under inspection. Our framework also measures the correctness of M2M transformation without having to transform them into any other formalism or to abstract any of their features. Naturally, we will continue our research focusing on the tool's scalability and cost (inspection effort) as well as defining weights to classify the severity of the founded errors.

References

1. Rahim, L., Whittle, J.: A survey of approaches for verifying model transformations. Softw. Syst. Model. 14, 1003–1028 (2015).
2. Stürmer, I., Conrad, M., Fey, I., Dörr, H.: Model Transformation Testing Challenges. In: Int. Workshop on Soft. Engineering for Automotive systems (SEAS '06). pp. 45–52 (2006).

3. Küster, J.M., Heckel, R., Engels, G.: Defining and validating transformations of UML models. In: Proceedings - 2003 IEEE Symposium on Human Centric Computing Languages and Environments, HCC 2003. pp. 145–152 (2003).
4. Granda, M.F., Condori-Fernandez, N., Vos, T.E.J., Pastor, O.: Towards the automated generation of abstract test cases from requirements models. In: 1st International Workshop on Requirements Engineering and Testing. pp. 39–46. IEEE, Karlskrona, Sweden (2014).
5. Granda, M.F., Condori-fernández, N., Vos, T.E.J., Pastor, Ó.: CoSTest : A tool for Validation of Requirements at Model Level. In: 25th International Requirements Engineering Conference - Tool Demo (2017).
6. Stürmer, I., Conrad, M., Fey, I., Dörr, H.: Experiences with model and autocode reviews in model-based software development. Proc. 2006 Int. Work. Softw. Eng. Automot. Syst. - SEAS '06. 45 (2006).
7. Van Amstel, M.F., Lange, C.F., Brand, M.G.: Using metrics for assessing the quality of ASF+SDF model transformations. In: 2nd International Conference on Theory and Practice of Model Transformations. pp. 239–248 (2009).
8. Amstel, V., Van Den Brand, M.G.J.: Model Transformation Analysis: Staying ahead of the maintenance nightmare. In: 4th Int. Conf. on Theory and Practice of Model Transformations. pp. 108–122 (2011).
9. Van Amstel, M., Bosems, S., Kurtev, I., Pires, L.F.: Performance in Model Transformations: Experiments with ATL and QVT. Lect. Notes Comput. Sci. 6707, 198–212 (2011).
10. Saeki, M., Kaiya, H.: Measuring Model Transformation in Model Driven Development. In: Int. Conference on Advanced Information, Systems Engineering. pp. 77–80 (2007).
11. Vignaga, A.: Metrics for Measuring ATL Model Transformations, Tech. rep., Universidad de Chile. (2009).
12. Kapová, L., Goldschmidt, T., Becker, S., Henss, J.: Evaluating maintainability with code metrics for model-to-model transformations. In: 6th Int. conference on Quality of Software Architectures: research into Practice–Reality and Gaps, QoSA'10. pp. 151–166 (2010).
13. Rahimi, S., Lano, K.: Integrating goal-oriented measurement for evaluation of model transformation. In: Int. Symp. on Computer Science and Soft. Eng. pp. 129–134 (2011).
14. Varró, D., Pataricza, A.: Automated Formal Verification of Model Tranformations. In: Critical Systems Development in UML. pp. 63–78 (2003).
15. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Sci. Comput. Program. 72, 31–39 (2008).
16. OMG: MOF 2.0/XMI Mapping Specification, v2.1, https://www.omg.org/spec/XMI/2.1/.
17. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. , Amsterdam (2008).
18. Object Management Group: OCL : Object Constraint Language. (2014).
19. España, S., González, A., Pastor, Ó.: Communication Analysis: A Requirements Engineering Method for Information Systems. In: 21st International Conference on Advanced Information Systems Engineering. pp. 530–545 (2009).
20. Tort, A., Olivé, A.: Case Study: Conceptual Modeling of Basic Sudoku, http://guifre.lsi.upc.edu/Sudoku.pdf.