A Qualitative Study of Model Transformation Development Approaches: Supporting Novice Developers

Gabriel Costa Silva, Louis M. Rose, and Radu Calinescu

Department of Computer Science, University of York,
Deramore Lane, York YO10 5GH, UK
gabriel@cs.york.ac.uk,
{louis.rose,radu.calinescu}@york.ac.uk

Abstract. Developing model transformations is not a straightforward task. It is particularly challenging when the developer has limited or no experience in this area. This not only impedes the adoption of model transformations, but also prevents companies from the benefits of Model-Driven Engineering. We qualitatively analyse eight of the most relevant approaches to developing model transformations cited in the literature. Different from most studies in this area, we focus on life-cycle activities other than implementation. By highlighting the strengths and weaknesses of these approaches, we help new developers in selecting an approach and complement existing studies in this area.

Keywords: Model transformation; Development; Analysis; Comparison

1 Introduction

The use of Model Transformations (MT) in software engineering leads to several benefits, such as complexity reduction and portability [3], [4]. However, developing MT is a challenge. As Siikarla et al. state in [12], "*An application developer with proper domain knowledge can manually create a target model based on a source model. He can probably even come up with some of the most often used rules of thumb for the transformations. However, defining a complete mapping from all possible source models to target models is remarkably harder.*" The state-of-the-art for developing MT consists of describing the MT definition informally in natural language [5], based on an "educated guess" [12], and implementing the transformation definition using a transformation language [6], [18]. This informal approach leads to several drawbacks [6], [18], [20], such as inconsistency between specification and implementation [5] and cost increase [8].

Since MT are software [12], they need to be engineered as software [6], using a systematic process of:

(i) specifying problems and requirements [5], [6], [7];
(ii) acquiring knowledge about semantic correspondences [19], [20];

(iii) modelling source and target models to outline a transformation [7], [12], [19];
(iv) creating mappings between meta-models [6], [19], [20];
(v) setting up constraints and rules for model mappings [6], [19], [20];
(vi) defining transformation architecture [6];
(vii) implementing the transformation definition using a transformation language [5], [6], [12], [20]; and
(vii) validating the transformation definition [5], [6], [12].

Furthermore, analogous to software development, MT can take advantage of further techniques to support its development, such as transformation patterns [7], [18]. As Guerra et al. advocate in [6], developing MT "*requires systematic engineering processes, notations, methods and tools*". Although there are several tools for implementing MT definitions [20], there is: (i) little work addressing the whole life-cycle of MT [12], [19]; (ii) lack of guidelines for the systematic development of MT, in particular, to support novice MT developers; and (iii) different perspectives regarding the best way to develop MT.

The analysis in this paper aims to contribute to the adoption of Model-Driven Engineering by providing a qualitative analysis of state-of-the-art approaches in developing model-to-model (M2M) transformations, investigating their strengths and weaknesses, and highlighting lessons learned for supporting MT developers starting in this area. Our analysis was prompted by the following research question: Which methods and techniques should a novice MT developer use? Unlike most work in MT, such as that reported in [10], our focus lies in activities of MT life-cycle other than implementation. In particular, we investigate processes, a modelling language, and transformation patterns to support MT development.

Note that this is not the only challenge associated with MDE adoption. However, challenges such as model and meta-model development are outside the scope of our paper. By a literature review, we identify approaches to support MT life-cycle. Then, we qualitatively analyse recent approaches according to three aspects (Section 2): (i) model transformation foundations; (ii) features; and (iii) applicability. Finally, we summarise lessons learned during our experience with analysing and adopting MT in the cloud computing domain (Section 3).

2 Analysis

In our investigation of M2M transformation development approaches, we identified no study, either qualitative or quantitative, comparing approaches to MT development in phases other than implementation. Furthermore, taking into consideration approaches we analysed in this paper, apart from the approach presented in [6], no approach was extensively evaluated, providing only worked examples to support a feasibility analysis. Moreover, so far, no approach for MT development has been widely adopted. These three facts make hard to choose a suitable approach when developing MT, fostering the concept of "*ad hoc*" MT development [6]. It becomes even harder when the MT developer has limited experience in this activity. In order to support novices in the task of selecting an

approach for MT development, we provide in this section a qualitative analysis of approaches present in the MT literature. Table 1 summarises our analysis.

2.1 Study Design

As Sjoberg, Dyba & Jorgensen define in [16], "*qualitative methods collect material in the form of text, images or sounds drawn from observations, interviews and documentary evidence, and analyze it using methods that do not rely on precise measurement to yield their conclusions.*" To this end, we used 13 criteria for analysing the approaches covered in our analysis, classifying the approaches into three groups: MT foundations, main features, and applicability of each approach.

The first group consists of foundation concepts of MT, as presented in [3] and [4]. The second group is based on the most relevant features implemented by approaches. Finally, the last group consists of our evaluation of the applicability of these approaches, taking into consideration the perspective of a novice MT developer. The evaluation of these approaches was exclusively based on information presented in their respective papers. As Seaman advocates in [11], by conducting this qualitative analysis we aim at providing rich and informative analysis to support novice MT developers. In addition, we aim at increasing the knowledge on how these approaches can contribute to designing MT.

Investigating model transformation literature, we identified eight research papers presenting approaches supporting M2M transformation development. As in our previous investigation [14] the IEEE digital library provided the most significant set of primary studies for our research, we decided to use this library as our primary source. Our search identified 121 papers, from which we critically analysed both title and abstract, resulting in two papers selected for full reading [7], [8]. In addition, we were supported by an MDE expert, who suggested another paper [6]. Finally, by analysing references and citations of previously selected papers, we found five more relevant papers [5], [12], [18], [19], [20].

Based on an analogy with software development, we classified the identified approaches as:

(i) *traditional* [6], [12] when they advocate an iterative process of refinements, from requirements to MT implementation and test;
(ii) *by example* [18], [19], [20] when they concentrate on simplifying the transformation development, focusing on model mappings [9];
(iii) *emergent* [5] when based on emergent software development approaches (e.g., agile), which proposes innovative ways to develop software; and
(iv) *transformation patterns* [7], [8] which enable the identification of recurrent relations in a model transformation, supporting the definition of transformation rules in a reusable way [8].

Table 1. Summary of key characteristics of MT approaches

		Traditional		By-Example			Emergent	Transformation Pattern	
		Siikarla et al. (1)	Guerra et al. (2)	Varró (3)	Wimmer et al. (4)	Sun, White & Gray (5)	Giner & Pelechano (6)	Jin & Guisheng (7)	Iacob, Steen & Heerink (8)
MT Foundations	Can it have multiple sources/targets?	No	Yes	No	No	No	No	No	No
	Can it support exogenous/ endogenous transformations?	Exogeneous	Both	Exogeneous	Exogeneous	Endogeneous	Exogeneous	Exogeneous	Both
	Can it support to mappings between source and target elements?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Can it support rule definitions?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Features	Is it language independent?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
	Which phases of transformation life cycle does it cover?	3, 4, 5, 7, 8	1, 2, 3, 4, 5, 6, 8	3, 4, 5, 7, 8	3, 4, 5, 7	3, 4, 5, 7, 8	1, 4, 5, 7, 8	4, 5	4, 5
	Where does the focus lies in?	Model	Both	Model	Model	Model	Model	Meta-model	Both
	How many types of artefacts does it specify?	4	9	1	3	3	1	1	1
	Does it requires tooling support?	No	Yes	Yes	Yes	Yes	Yes	No	Yes
Applicability	How sound is the approach?	Minimal	Excellent	Minimal	Minimal	Minimal	Minimal	Minimal	Good
	Which type of evaluation was carried out?	Feasibility	Case study	None	None	None	Feasibility	Feasibility	Feasibility
	How much detail is provided?	Good	Excellent	Good	Excellent	Excellent	Minimal	Minimal	Good
	Can support other approach?	No	Yes	No	No	No	No	Yes	Yes

2.2 Model Transformation Foundations

The first group of criteria consists of: multiple source/ target, exogenous/ endogenous MT, mappings, and rules. It is critical to note that the papers we analysed provided little, or no clear information about the two first criteria. For those cases, we analysed the examples provided throughout the paper as the main reference to infer answers for this question. To simplify the reference to each approach, Table 1 enumerates approaches using numbers from (1) to (8). These numbers will identify their respective approaches from now on.

Taking into consideration the support for multiple source/ target, only (2) explicitly stated the support for multiple source and targets by their mapping diagram. All other approaches suggested support for only single source/ target models. For example, (3) made clear the decision taken by a set of stated assumptions. Siikarla, (1), suggested his decision in a figure, used to explain his approach. Analysing the transformation patterns presented, none of them consider multiple source/ targets. For all other approaches, they suggested their decision by examples presented throughout the paper.

Regarding the type of transformation supported, most of approaches support only exogenous transformations. The only exception is (5), which presented only endogenous examples, suggesting that this approach aims at also supporting endogenous transformations. Examples presented by (2) suggested that it supports

both types of transformations. Likewise, the patterns flattening and mapping, presented by (8), indicated that it supports both types of transformations.

As mappings and rules are core concepts in MT, both concepts are considered by all analysed approaches. The main difference in this regard lies in how these concepts are implemented by different approaches. For example, whereas (1) set up correspondence examples to map models, and transformation patterns to map meta-model, (6) specified mappings by test cases. Regarding rules, whereas (2) defined it in their low-level design, (3) proposed their automatic generation.

2.3 Features

The second group of criteria consists of: language independence, phases covered, focus, artefacts, and tooling support. In the context of this paper, language independence means that the approach is not specific to a particular language for the source and target models/meta-models. Apart from (8), that defines its transformation patterns in the context of QVT, all other approaches are language independent – though the examples presented are based on a particular language, such as (1), that adopted UML. To analyse the phases of MT life cycle covered by approaches, we took into consideration the general process for developing MT, introduced in Section 1. The set of phases we included in this process is the result of an analysis of MT literature. Table 2 summarises the phases covered by each approach, and how each approach covers such phase.

Regarding the focus of each approach, although MT is defined at the meta-model level, most approaches focus on the model rather than the meta-model level. As Wimmer et al. explain in [20], meta-models might not define all language concepts explicitly. Regarding the approaches we analysed, the exception is (7), which focuses on the meta-model, and both (2) and (8), which address both model and meta-model. It is important to note that (8) described their patterns in terms of model, but the definition is made at meta-model level.

Approaches proposed a number of artefacts to support the developer. Table 1 shows the number of artefact types though it is central to note that this information is unclear in some papers, such as those which present (6), (4), and (5). Finally, regarding tool support, (1) and (7) do not require a particular tool. In the context of this analysis, requiring tool support means that the approach cannot be wholly applied without a particular tool. For example, (7) does not require a particular tool although it defines several automatic activities. Likewise, although (2) defines a family of languages, its MT development process is language independent. Unlike the other approaches, (2) generates automatically only test and traceability artefacts. In particular, by-example approaches define semi-automatic tasks, that require specific tools. Approach (6) also defined the use of specific tools, such as HUTN, EVL, and an adapted version of JUnit.

2.4 Applicability

To evaluate the applicability of each approach, we considered four criteria: soundness, evaluation, level of technical detail, and the ability of the analysed approach

Table 2. MT phases covered by each approach analysed

	Traditional		By-Example			Emergent	Transformation Pattern	
	Siikarla et al. (1)	Guerra et al. (2)	Varró (3)	Wimmer et al. (4)	Sun, White & Gray (5)	Giner & Pelechano (6)	Jin & Guisheng (7)	Iacob, Steen & Heerink (8)
1 - Specifying problems and requirements		Requirements diagram				Test case		
2 - Acquiring knowledge about semantic correspondences		Formal specification/ Scenario definition						
3 - Modelling source and target models to outline a transformation	Correspondence examples	Scenarios definition	Step 1 - Mapping models	Step 1 - Creating models	Demonstration			
4 - Creating mappings between meta-models	Transformation pattern/ definition	Mapping diagram	Step 1, 2 - Derive transformation rules	Step 2 - Creating correspondence	Automatically infered	Test case	Specified by patterns	Specified by patterns
5 - Setting up constraints and rules for model mappings	Transformation pattern/ definition	Rule diagram	Step 2, 3 - Rules adjustment	Step 2, 3 - Deriving correspondences	Automatically infered	Test case	Specified by patterns	Specified by patterns
6 - Defining transformation architecture		Architecture diagram						
7 - Implementing the transformation definition by a transformation language	Transformation implementation		Step 4 - Rules execution	Step 3 - Deriving correspondences	Step 5 - Replaying transformations and checking	Transformation language		
8 - Validating the transformation definition	Evaluation	Test language	By process iteration		Step 5 - Replaying transformations and checking	Comparing result of transformation with test case		

to support another — like design patterns can support software development process. We define the soundness of an approach, using the following scale: (i) *minimal* — the paper introducing the approach provides very limited information about approach theoretical foundations; (ii) *good* — the paper justifies the decisions taken and reasons to underpin their decisions, even the information is limited; and (iii) *excellent*, meaning that the text not only describes decisions and their reasons, but also provides empirical evidence and basis from literature.

Taking into consideration this scale, only (2) was classified as having an *excellent* soundness whereas (8) was classified as having a *good*. This result becomes worse when analysed along with the next criterion, evaluation. None of by-example approaches were evaluated in the papers analysed. Apart from (2), which conducted two case studies, other approaches conducted feasibility analyses, taking into consideration worked examples. These two results highlight the need for empirical evidence to support future work in this area. This is particularly important for the industrial adoption of these approaches. However, note that these results do not invalidate the approaches. In fact, as discussed in Section 3, many of these ideas are valid and useful.

Aiming at the application of these approaches, we evaluated the level of technical detail provided in the paper. There are three possible classifications for this criterion: (i) *minimal*, when the text does not provide enough information to support the application of this approach; (ii) *good*, when the text provides enough information to support the application of this approach, though detailed information might be missing; and (iii) *excellent*, when the text not only provide

enough information, but also further details about activities, such as examples. In this regard, (7) and (6) were classed as *minimal*. Approaches (2), (4), and (5) provide *excellent* information, supported by examples that complement the understanding. However, it is critical to point out that these three approaches rely on tooling support, and the knowledge regarding the tools might be not enough. The other three approaches were classified as *good*.

Finally, we analysed whether these approaches could support other approach. For example, a traditional software development process might be supported by several approaches, such as design patterns and graphical modelling languages. Apart from transformation patterns and the MT language (approach (2)), which by definition could be applied to support other approaches, all other approaches defined their particular, non-extendable, life-cycle. In some cases, such as that of (5), the need for tooling support creates additional dependency.

3 Lessons Learned

In this section, we summarise lessons learned while transforming a cloud model [13], defined as part of our approach to support cloud portability [15], into a TOSCA definition. Cloud computing is a computing model in which resources, such as computing, are provided as services through the Internet [1]. Topology and Orchestration Specification for Cloud Applications (TOSCA) is a standard specification supported by OASIS to enable cloud portability [2]. This section is based on our attempt to systematically apply the approaches previously presented. Lessons reported in this section complement the analysis in Section 2.

Developing model transformations is as complex as traditional software development

Comparing the generic MT process introduced in Section 1 with the traditional software development (TSD) life-cycle, such as waterfall [17], we can conclude that both processes are similar. A developer needs requirements to define the objectives, artefacts to guide the development, and tests to check for errors. Like the TSD, the code is the main artefact, which represents the MT definition. However, a number of questions arise when starting the requirement definition for a MT. In contrast to TSD, in which one defines several requirements, in MT, inicially, there is one single requirement: transform model A into model B.

In MT, the single requirement proposed must be broken down into several others, specifying that the entity X, in the meta-model A, will become the entity Z, in the meta-model B. To this end, the requirements diagram presented in [6] is a relevant contribution since it enables requirement decomposition and the creation of links between requirements to set up dependencies. However, to define such a mapping, it is necessary to have a clear understanding of semantic correspondences between the meta-models involved. Furthermore, unless these semantic correspondences have been tested before, establishing the requirements would be an error-prone task. For example, when we defined the requirements

for our cloud-to-TOSCA MT, we did not know which TOSCA entity a cloud entity will become – though we knew very well both domains.

Thus, we had to analyse semantic correspondences of both meta-models before defining the requirements. In addition, we had to test if these correspondences made sense. In this regard, the test-driven approach proposed in [5] was useful. A test-case is an artefact which outlines these semantic correspondences. Then, by implementing the transformation, this initial assumption is confirmed or refuted. However, from a novice MT developer, the complexity involved in defining requirements for MT is far harder than in TSD. In addition, requirement definition and mapping definition are two close activities. Finally, M2M transformations might involve model-to-text transformations as well, making the process even more complex. Thus, despite the similarity with TSD, in our perspective, MT development requires extra artefacts and activities.

Models work well as examples, but not as the main transformation drivers

By-example approaches, in particular, advocate the use of models rather than meta-models as the main driver of a MT. Indeed, having two models, one representing a domain A, and another representing a domain B, aids the design of an A-to-B MT. However, creating these models is not trivial. Although it might be an intuitive process when creating a transformation from a UML class diagram to an E-R diagram – a common example used in the literature, other domains require a huge effort. In our case, we found it to be impossible to devise a TOSCA model based on our cloud model without an in-depth preliminary analysis of semantic correspondences. The reason for that is quite simple: a model conforms to a meta-model. Therefore, one cannot create a representation of model A, which conforms to meta-model A, in conformance with meta-model B unless the semantic correspondences between meta-models A and B are known beforehand.

For example, the by-example approaches proposed in [19] and [20], define in their first activities the creation of source and target models, and the mapping of entities between these models. In our case, we already had the cloud model, however, we expected to follow a well-defined MT process in deriving the TOSCA model (target). At that moment, we could infer that a cloud *Service* is similar to a TOSCA *TNodeType*, but we did not know correspondences for other entities, such as a cloud *Region*, and *User*. Thus, a process advocating the mapping of models to achieve meta-models correspondences was not applicable because without the meta-model correspondences it was impossible to derive the models.

In this regard, we learned that by-example approaches could be useful when meta-model correspondences are already known, and two well-known meta-models are given. Thus, models can be mapped and meta-model correspondences can be automatically generated using these approaches. On the other hand, the creation of two models is quite useful when designing MT as a way to validate meta-model mappings. For example, after identifying that a cloud *Resource* is equivalent to a TOSCA *TNodeType*, we created a TOSCA model representing this mapping. Then, we could validate the model in two ways: checking in the general context

of TOSCA whether this transformation makes sense, and submitting the generated model to a TOSCA runtime environment. If the environment could process the specification, it meant that the transformation succeeded.

Other lessons

- Although not yet addressing all MT development concerns, the analysed approaches provide very useful contributions. Overall, we concluded that, like MT area, the identified approaches are still maturing. As shown by Table 1, most of them were neither extensively evaluated nor sound enough. However, they provide several insights about performing MT, such as correspondence examples [12], requirements diagram [6], and test-cases [5];
- Testing is critical in MT as it is in TSD. It is important to carry out several tests when developing MT. In our experience, we identified problems with different datatypes (e.g., conversion of *String* to *xs:string*), names (space between nouns versus no space), and one entity in the source meta-model being mapped to several others in the target meta-model;
- Capturing trace links between source and target model elements is a good practice for MT, particularly if a MT becomes complex. In our experience, one single entity in the cloud meta-model became three entities in the TOSCA meta-model, which in turn gave rise to several others. At the end, the set of dependencies created was so complex, that it was hard to validate them. Inspecting the trace model can help considerably in cases like this, as it enables to identify source and target entities.

4 Conclusion

In our investigation of M2M transformation development, we identified no study, either qualitative or quantitative, comparing approaches for MT development in phases other than implementation. This complicates the selection process of a suitable approach when developing MT, fostering the concept of "ad hoc" MT development. It becomes even harder when the MT developer has limited experience in this activity. To support the selection of an approach for MT development, we provided in this paper a qualitative analysis of eight state-of-the-art approaches for MT development. This analysis took into consideration 13 criteria, classified in three groups: model transformation foundations, features and applicability. We complemented this analysis presenting the lessons learned from our own experience with developing a MT for cloud domain.

Acknowledgments. This work was funded in part by CNPq - Brazil.

References

1. Armbrust, M., Stoica, I., Zaharia, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A.: A view of cloud computing. Communications of the ACM 53(4), 50–58 (Apr 2010)

2. Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: TOSCA: Portable Automated Deployment and Management of Cloud Applications. In: Advanced Web Services, chap. TOSCA}: Po, pp. 527–549. Springer, New York (2014)
3. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice. Morgan & Claypool Publishers (2012)
4. France, R., Rumpe, B.: Model-driven Development of Complex Software: A Research Roadmap. In: Future of Software Engineering (FOSE '07). pp. 37–54. IEEE, Minneapolis, MN (May 2007)
5. Giner, P., Pelechano, V.: Test-Driven Development of Model Transformations. In: MDE Languages and Systems, pp. 748–752. Springer, Berlin (2009)
6. Guerra, E., de Lara, J., Kolovos, D.S., Paige, R.F., dos Santos, O.M.: Engineering model transformations with transML. Software & Systems Modeling 12(3), 555–577 (2013)
7. Iacob, M.E., Steen, M.W.A., Heerink, L.: Reusable Model Transformation Patterns. In: 2008 12th Enterprise Distributed Object Computing Conference Workshops. pp. 1–10. IEEE, Munich (Sep 2008)
8. Jin, L., Guisheng, Y.: Method of constructing model transformation rule based on reusable pattern. In: 2010 International Conference on Computer Application and System Modeling (ICCASM 2010). pp. 519–524. IEEE, Taiyuan (Oct 2010)
9. Kappel, G., Langer, P., Retschitzegger, W., Schwinger, W., Wimmer, M.: Model Transformation By-Example: A Survey of the First Wave. In: Düsterhöft, A., Klettke, M., Schewe, K.D. (eds.) Conceptual Modelling and Its Theoretical Foundations, pp. 197–215. Springer Berlin Heidelberg, Berlin (2012)
10. Lano, K., Kolahdouz-Rahimi, S., Poernomo, I.: Comparative Evaluation of Model Transformation Specification Approaches. International Journal of Software and Informatics 6(2), 233–269 (2012)
11. Seaman, C.B.: Qualitative Methods. In: Guide to Advanced Empirical Software Engineering, pp. 35–62. Springer London, London (2008)
12. Siikarla, M., Laitkorpi, M., Selonen, P., Systä, T.: Transformations Have to be Developed ReST Assured. In: Vallecillo, A., Gray, J., Pierantonio, A. (eds.) Theory and Practice of Model Transformations, pp. 1–15. Springer, Berlin (2008)
13. Silva, G.C., Rose, L., Calinescu, R.: Cloud DSL: A Language for Supporting Cloud Portability by Describing Cloud Entities. In: 2014 CloudMDE Workshop. p. To be published. Valencia (2014)
14. Silva, G.C., Rose, L.M., Calinescu, R.: A Systematic Review of Cloud Lock-In Solutions. In: 2013 IEEE CloudCom. pp. 363–368. IEEE, Bristol (Dec 2013)
15. Silva, G.C., Rose, L.M., Calinescu, R.: Towards a Model-Driven Solution to the Vendor Lock-In Problem in Cloud Computing. In: 2013 IEEE CloudCom. pp. 711–716. IEEE, Bristol, UK (Dec 2013)
16. Sjoberg, D.I.K., Dyba, T., Jorgensen, M.: The Future of Empirical Methods in Software Engineering Research. In: FOSE '07. pp. 358–378. IEEE, Minneapolis, MN (May 2007)
17. Sommerville, I.: Software Engineering. Addison Wesley, Harlow, 8 edn. (2007)
18. Sun, Y., White, J., Gray, J.: Model Transformation by Demonstration. In: Model Driven Engineering Languages and Systems, pp. 712–726. Springer, Berlin (2009)
19. Varró, D.: Model Transformation by Example. In: Model Driven Engineering Languages and Systems, pp. 410–424. Springer, Berlin (2006)
20. Wimmer, M., Strommer, M., Kargl, H., Kramler, G.: Towards Model Transformation Generation By-Example. In: HICSS'07. pp. 285–294. IEEE, Waikoloa (2007)