Motrusca: interactive model transformation use case repository*

Joost van Pinxten[1] and Twan Basten[1,2]

[1] Eindhoven University of Technology, Eindhoven, the Netherlands
[2] TNO-ESI, Eindhoven, Netherlands
{j.h.h.v.pinxten,a.a.basten}@tue.nl

Abstract. Modeling and model transformations tools are maturing and are being used in larger and more complex projects. The advantage of a modeling environment and its transformation tools cannot be easily exploited by non-expert users as many subtle intricacies determine the efficiency of transformation languages and their tools. We introduce transformation use case examples that highlight such language/tooling properties. These simple, non-trivial examples have been extracted from an experiment with transformations of Design Space Exploration models. These examples show some typical modeling patterns and we give some insight how to address the examples. We make a case for initiating an interactive, on-line repository for model transformation use cases. This repository is aimed to be example-centric and should facilitate the interaction between end-users and tooling developers, while providing a means for comparing the applicability, expressivity, and efficiency of transformation tools.

Keywords: model transformations, comparison, design space exploration

1 Introduction

Model transformations play a crucial role in Model-Driven Development (MDD) since the OMG Meta-Object Facility [14] has been introduced in 2005. Amongst others, the Eclipse Modeling Framework has eased the creation of fully-featured editors (with Xtext and Sirius) for Domain-Specific Languages (DSLs). This has greatly increased the popularity of MDD in the past years. With increasing maturity of MDD environments [15] one would also expect a similar uptake in usage of model transformation tools. Several model transformation languages and tools have been introduced since 2006, such as QVTo [6], Epsilon [13], and ATL [9] for Eclipse, but also MetaEdit+ MERL [10] and GME's GReAT [2]. However, we find that there has been limited success in this area.

From our experiences, it is difficult to make an informed decision on what the best approach is for certain model transformation scenarios. Typically, the developer's experiences with other programming languages and personal preference

*This work was carried out as part of the Octo+ project with Océ Technologies B.V. under the responsibility of TNO-ESI, Embedded Systems Innovation by TNO.

dictate the choice for a particular tool/language. The high level of abstraction makes it difficult to assess the differences between tools and languages, as only basic documentation and examples are available. This information typically focuses only on the strong aspects of a single tool or language[3] [4].

We therefore recognize the lack of and need for an up-to-date unified public repository for model transformation challenges. To make an informed decision, we need clear, non-trivial use cases with motivating scenarios and corresponding example implementations expressed in several model transformation languages.

Throughout this paper, we use the Design Space Exploration (DSE) scenario as context. DSE is a critical aspect in the design of Embedded Systems as early design decisions can be supported by efficient simulation and analysis of high-level models. This can shorten the time-to-market and reduce late detection of potential performance problems.

We first introduce the motivation, goals, and core ideas behind the Motrusca interactive repository for model transformation use cases (Sec. 2). We then introduce an example motivating scenario (Sec. 3) from which we highlight some example use cases and motivate why they are hard to express succinctly and efficiently in current model transformation tools (Sec. 4). We discuss related work in Sec. 5 and conclude in Sec. 6.

2 Interactive use case repository

With the plethora of model transformation languages and tools available, it has become hard to even enumerate all (Eclipse-based) transformation languages. It is even harder to determine which one can or should be used. It is often unclear what functionality a transformation tool has (or lacks), what the adoption rate is and what the current status of such tools is. Each transformation language comes with its own set of examples and therefore cannot be directly compared to another. Comparing transformation languages and choosing the right one is currently a non-trivial activity.

2.1 Motivation and goals

With the Motrusca interactive model transformation use case repository, we intend to change the way examples are created. Instead of reasoning from the capabilities of a language, we want to reason from the perspective of a use case. Each use case can then be implemented in different model transformation languages, or even different variations in a single model transformation language.

These use cases are the starting point for a repository of good practices in model transformations and discussions between model/transformation developers and tool developers. Although this information can also be captured on forums, StackOverflow[5], tutorials and blogs, we feel that a dedicated place to

[3] `www.eclipse.org/atl/atlTransformations/`

[4] `www.eclipse.org/epsilon/examples/`

[5] `www.stackoverflow.com`

model transformations is essential to ensure maximum usefulness and visibility of the information. The Transformation Tool Contest (TTC) [7] is a yearly academic event dedicated to a similar goal. However, Motrusca aims to complement their activities by creating continuous, interactive feedback from model transformation user groups.

In short, the Motrusca repository aims to achieve the following goals:

- identify use cases and best practices for model transformations
- trigger interaction between transformation developers and tool developers
- present alternative solutions to model transformation challenges
- provide a means to compare model transformation tooling/languages (e.g. on their efficiency and applicability)

The use cases described in this paper have been added to the repository, along with some initial implementation examples that show some of the differences between Epsilon Transformation Language (ETL), QVTo and ATL.

2.2 Identifying use cases and best practices

New use cases will first be reviewed before implementations can be submitted. Users should be motivated to supply enough information for a minimal example that can be used for discussion. A proper use case example must have at least:

- a user story motivating the context (a scenario)
- a minimal example (meta)model
- a description of what the use case addresses and why the use case is interesting to discuss

Motrusca will aim at making the processes of providing feedback and information as easy and valuable as possible. Transformation developers can then more easily share their experiences and expertise. It gives tool developers insight in the way users think the tool should be used and also leads to exposing novice users to the best practices. Inspired by test-driven development best practices, fallacies of one model transformation tool should be documented and consequently avoided by other tools. We are essentially building a set of generic integration tests that can be leveraged by transformation language/tool developers.

2.3 Interaction between tool users and developers

By providing a view from the perspective of a use case instead of from a tool/language, novice users can become more comfortable with the best practices faster and might be able to choose the proper tooling without the need for much experimentation. One can compare this model to the popular StackOverflow model, where users can ask questions and the community can provide answers and ask questions to clarify the questions. Motrusca will initially aim at Eclipse-based model transformation languages, but the ideas should translate to other modeling platforms as well. Users will be able to vote for the answer/solution that they deem most appropriate to the use case at hand, similar to questions on StackOverflow.

2.4 Providing alternative solutions

The complexity of the model transformation does not have to originate from the transformation language; for example, the source or target meta-model may be inconvenient or inefficient for transformation purposes. Motrusca users are therefore encouraged to provide out-of-the-box thinking and contribute insight into different realizations (of the meta-model) and show the effect on the model transformation use case.

2.5 Comparison overview

It is important that the solutions to the use cases can be compared to one another. A comparison on several aspects based on our motivating case is shown in Fig. 1. It shows several aspects similar to the quality framework of Kolahdouz-Rahimi et. al. [12] to rate the transformation solutions that we have implemented in Java, IncQuery + Xtend and Epsilon. Motrusca users will be motivated to provide feedback in terms of these kind of quality attributes. By aggregating such information, Motrusca can automatically indicate the popularity and effectiveness of transformation tools.

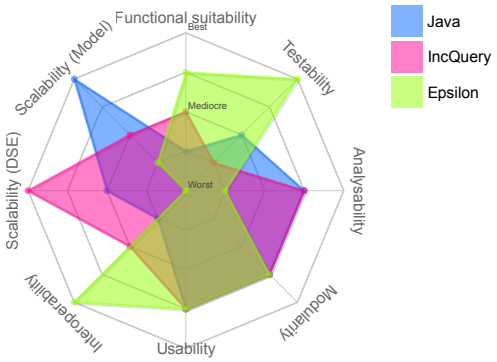

Fig. 1: Overall comparison of model transformation approaches in DSE

Certain use cases are primarily related to the performance efficiency of the model transformations, which can be properly quantified. The performance can be measured, by making standardized experiments available through a cloud-based service such as Travis[6] or possibly through SHARE [18]. In addition to performance efficiency, the scalability of the runtime is sometimes of interest to be investigated and model instances that reflect scalability issues can be provided with the use cases.

[6] `www.travis-ci.org`

3 Motivating scenario

The Octopus tool set [4] is developed to support the DSE process that serves as the motivating case study in this paper. To facilitate automatic DSE in the tool set, a Design Space Exploration Intermediate Representation (DSEIR [3]) has been introduced. This formal representation functions as an intermediate between DSLs and a variety of analysis tools (e.g. CPN Tools [8], UPPAAL [5], SDF3 [16]). The use of an intermediate format facilitates re-use of models and tools, and improves model consistency and code maintainability. The Octopus tool set provides generic tools for coping with design decisions in all kinds of embedded systems. The following subsections describe the motivating industrial use case, the philosophy of DSE with Octopus, and the high-level overview of model transformations in Octopus.

3.1 Copier data paths

The primary motivating use case for the Octopus tool set is the DSE for cost-effective software/hardware configurations for the image processing data path in high-performance, high-volume copier applications (Océ Technologies B.V.[7]). The DSE supports early design decisions where, for example, the type of algorithms and number of processors are explored.

In this domain, the software as well as the execution platform are modeled. The software is modeled with tasks (e.g. scanning, image processing steps, printing) that communicate data and require computation and storage services.

3.2 Design Space Exploration with Octopus

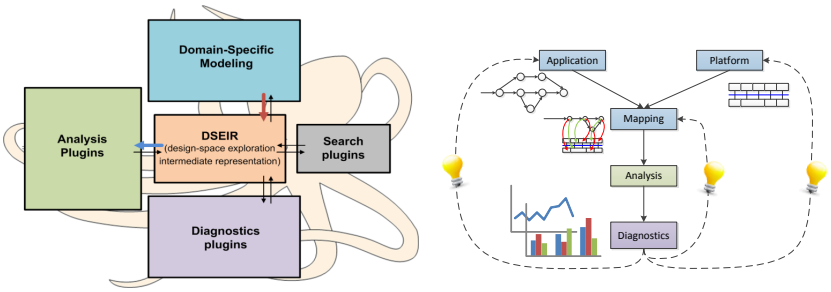

Fig. 2: Separation of concerns in Octopus (from [4]) (left) and Y-chart; separation of platform, application, mapping and diagnostics [11] (from [3]) (right)

[7] www.oce.com

The Octopus tool set is designed as a modular system (Fig. 2, left) for DSE of software-intensive embedded systems. It provides analysis and simulation services for DSLs that translate to DSEIR. Model transformations play a crucial role in both the analysis plugins and domain specific modeling front ends. The Octopus tool set already supports several analysis/simulation engines, by transformations from DSL to DSEIR and DSEIR to analysis/simulation models.

The model transformations to the analysis plugins are complex mappings of the DSEIR concepts onto different sets of concepts. The transformation from domain specific modeling front end imposes a different requirement; it should support the modeling engineer by providing high-quality feedback on analysis results in terms of the original model components.

3.3 Transformation flow in Design Space Exploration

Octopus contains several modeling front ends that allow the definition of parameterized DSEIR models. These models can be described in terms of a parameterized mapping, application, and platform as indicated in Fig. 2 (right). An application contains tasks, which require a specific service in order to be executed. The platform consists of resources that can provide a service at some rate. The mapping specifies that the required services are allocated on a resource that provides them. The Octopus tool set uses an experiment definition DSL (DseirText) to indicate the design space by stating the (range of) values for the parameters, the model, and type of analysis or simulation to be executed.

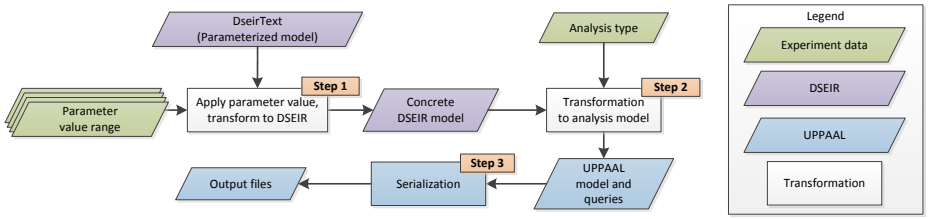

Fig. 3: Octopus example transformation orchestration

The transformation orchestration for DSE with the Octopus tool set is depicted in Fig. 3. It denotes three distinct transformation steps. Step 1 is specific to the modeling front-end, and steps 2 and 3 are specific to an analysis type. For the UPPAAL case, the experiment indicates that the UPPAAL engine should be used to analyze the latency bounds. The transformation engine needs to:

1. create concrete DSEIR models from domain-specific models; e.g. by substituting parameters by values and mapping DSL concepts to DSEIR concepts,
2. transform concrete DSEIR models to UPPAAL models,
3. serialize the UPPAAL model into UPPAAL model/query text.

The transformation from the input format to a DSEIR model consists primarily of copying the structure and binding values to expressions based on the parameter values that are provided in the experiment. The concrete DSEIR model is then transformed into an abstract UPPAAL model and query definition. Finally, the resulting model and queries are serialized into files that serve as the input for the UPPAAL engine.

4 Model transformation use cases

In this section, we highlight a few use cases that we have extracted from the DSE scenario, in particular from the scenario described in Fig. 3. We discuss briefly what difficulties arose when implementing these examples and how different model transformation languages could handle them. We focus on the following Eclipse-based model transformation tools: QVTo, Epsilon, and ATL. For more details on the use cases, as well as models and transformation implementations, see the Motrusca website `www.motrusca.net`.

4.1 Combining partial specifications into complete specifications

The separation of concerns in DSE inherently leads to multiple partial specifications that can be combined in several ways; Application, Platform, and Mapping (see Fig. 4) are reusable specification parts. Each combination of partial specifications leads to (at least) one concrete output model, which is compiled from the referenced partial specifications. This leads to the following requirements:

1. an arbitrary (unknown) number of target models may need to be produced
2. a set of source elements (e.g. the partial specifications) are transformed into a (set of) target element(s)
3. trace output model elements back to input elements

The first requirement cannot be specified properly in languages like QVTo, ETL, and ATL; the models to be generated need to be enumerated explicitly. We therefore need a parameterizable workflow that repeatedly calls the transformation to generate a specific output model from the input. Such a workflow can only be effective when a single transformation can be executed with a small processing overhead.

The second requirement, considering that multiple output models can be generated from a multiple sources, is inherently difficult to achieve in single-source rules as used in ETL. Before the introduction of EPL (Epsilon Pattern Language), it was impossible to define rules such that a target element could be traced back to the respective source elements. Transformation traces track source to target elements and are used to compute which element must be referenced. The third requirement is therefore an extension to the second requirement. QVTo allows mapping of additional source elements by passing these as parameters to the mapping function. ATL allows definition of multiple source elements in its transformation rules.

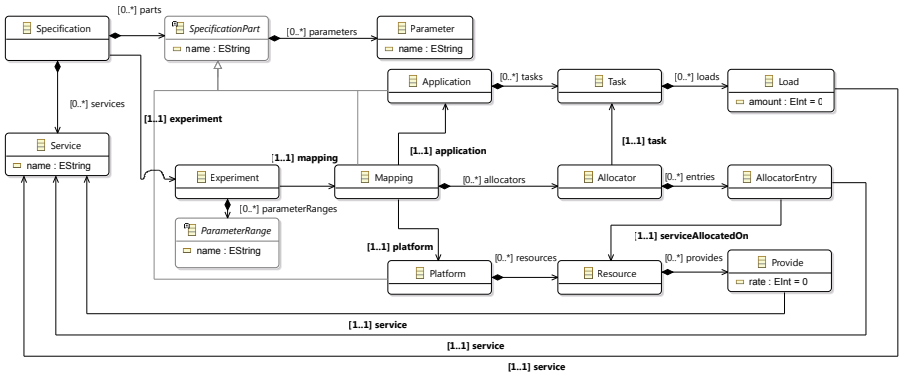

Fig. 4: Simplified excerpt from DseirText metamodel

4.2 Similar structures

Recall from Section 3.3 that the Octopus toolset defines experiments in a DSL DseirText, which are then exported to the intermediate format DSEIR. The DseirText and DSEIR metamodels in Eclipse are very similar; DseirText contains experiment definitions and parameters, whereas DSEIR does not. The translation between these model instances therefore consists of duplication of most of the structure, with concrete values for the parameterized DseirText expressions. In pure transformation approaches, this will lead to a lot of boilerplate code, as each concrete class in DseirText needs to be transformed into its direct concrete DSEIR counterpart. A higher-order transformation could describe these copy-actions more succinctly.

This problem has been partly addressed in ATL with a refining mode [17] [12] for such transformations. the resulting refined model may overwrite the source model, or can be saved in a new location. Epsilon Flock [13] is designed to migrate between different versions of a metamodel, but can also be applied to similar metamodels, to achieve a similar effect as the ATL refining mode.

4.3 Expanding parameters to concrete values

In DseirText, an Experiment contains a range of values for the parameters required in the SpecificationParts (see Fig. 4). As each SpecificationPart may be defined before any Experiment refers to it, the range of a parameter (ParameterRange) and the actual usage of a Parameter are decoupled. In Java, a simple mapping of parameter name to parameter value can be propagated and queried by expression tree visitors; this makes looking up parameter values inexpensive.

We have been unable to come up with a clean, concise and efficient way to express such transformations. With the Epsilon languages, we can leverage EPL in the ETL context to achieve a similar effect. However, this unnecessarily convolutes the definition of the transformation, as the underlying execution

mechanisms need to be instructed to cooperate, which is not a trivial task. In QVTo, it is possible to look up the defining ParameterRange for a certain encountered Parameter with the `->any(self.name = parameter.name)` operator. This has, however, linear time worst case behavior, in comparison to constant lookup time in the Java implementation.

4.4 Semantic gaps in expression trees

The basic concepts in the UPPAAL language are very different from the basic concepts in the DSEIR language and the expression trees transformation is therefore a complex mapping. Consider for example communication between two tasks; in the DSEIR concepts, task A can write directly into the input buffer of task B. However, in UPPAAL, there is no such communication possible and the communication is forced to use a global buffer.

This kind of semantic gap typically leads to many convoluted rules to capture the mapping logic. Expressions related to the buffers have to be considered in a different way than normal expressions; in procedural programming, a parameterized expression visitor can be used. It is, however, non-trivial to come up with a maintainable and efficient solution in model transformation languages.

5 Related work

A framework based on the 2001 ISO/IEC software product quality model has been defined for and applied to model transformation to quantify several aspects of model transformation approaches (language and tool combination) [12]. Our work complements this work.

There have been a few attempts at public metamodel [1] and model transformation[8] [7] repositories for gathering insight into usage and performance of transformation languages and tools. The yearly TTC records and compares expert results for a particular model transformation use case [7], providing insight into the status of state-of-the-art model transformation. The results are reported primarily in terms of functionality and appeal. In contrast to these repositories, Motrusca will enable much tighter interaction between transformation and tool developers. Motrusca also enables direct, interactive comparison of transformation languages.

6 Conclusions

In this paper, we have introduced a scenario and corresponding use cases that are hard to implement efficiently with the current state-of-the-art model transformation tools. We have highlighted several aspects that are hard for novices to distinguish without having to dive deep into a language. Taking industrial model transformation challenges as a starting point can boost the adoption and

[8] `www.eclipse.org/atl/atlTransformations/`

advancement of model transformation tools and languages. We introduce the goals of Motrusca as an interactive on-line repository and motivate its existence. The presented use cases are a good starting point for the Motrusca repository at `www.motrusca.net`.

References

1. The atlanmod zoo. `http://www.emn.fr/z-info/atlanmod/index.php/Zoos`.
2. Daniel Balasubramanian, Anantha Narayanan, Christopher van Buskirk, and Gabor Karsai. The graph rewriting and transformation language: Great. *Electronic Communications of the EASST*, 1, 2007.
3. T. et al. Basten. Model-driven design-space exploration for embedded systems: The octopus toolset. In *Proc. of ISoLA'10*, pages 90–105. 2010.
4. T. et al. Basten. Model-driven design-space exploration for software-intensive embedded systems - (extended abstract). In *Proc. of FORMATS'12*, pages 1–6, 2012.
5. G. Behrmann, A. David, K.G. Larsen, P. Pettersson, W. Yi, and M. Hendriks. Uppaal 4.0. In *QEST'06*, pages 125–126. IEEE Computer Society, 2006.
6. Object Management Group. Meta object facility (mof) 2.0 query/view/transformation, v1.1. 2011.
7. E. Jakumeit, S. Buchwald, D. Wagelaar, L. Dan, Á Hegedüs, M. Hermannsdörfer, T. Horn, E. Kalnina, C. Krause, K. Lano, M. Lepper, A. Rensink, L.M. Rose, S. Wätzoldt, and S. Mazanek. A survey and comparison of transformation tools based on the transformation tool contest. *SCP*, 85, Part A(0):41 – 99, 2014.
8. K. Jensen, L.M. Kristensen, and L. Wells. Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. *STTT*, 9(3-4):213–254, 2007.
9. F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev, and P. Valduriez. Atl: a qvt-like transformation language. In *Proc. of OOPSLA'06*, pages 719–720. ACM, 2006.
10. Steven Kelly and Juha-Pekka Tolvanen. *Domain-specific modeling: enabling full code generation*. John Wiley & Sons, 2008.
11. B. Kienhuis, E. Deprettere, K. Vissers, and P. van der Wolf. An approach for quantitative analysis of application-specific dataflow architectures. In *Proc. ASAP*, pages 338–349. IEEE, 1997.
12. S. Kolahdouz-Rahimi, K. Lano, S. Pillay, J. Troya, and P. van Gorp. Evaluation of model transformation approaches for model refactoring. *SCP*, 85:5–40, 2014.
13. D. Kolovos, L. Rose, A. García-Domínguez, and R. Paige. *The Epsilon Book*. February 2014.
14. OMG. ISO/IEC 25010 - Information technology – Meta Object Facility (MOF). Technical report, 2005.
15. R.F. Paige and D. Varró. Lessons learned from building model-driven development tools. *Software & Systems Modeling*, 11(4):527–539, 2012.
16. S. Stuijk, M. Geilen, and T. Basten. Sdf3: Sdf for free. In *Proc. of ACSD '06*, pages 276–278, 2006.
17. M. Tisi, S. Martínez, F. Jouault, and J. Cabot. Refining models with rule-based model transformations. Technical report, 2011.
18. P. van Gorp and S. Mazanek. Share: a web portal for creating and sharing executable research papers. *Procedia Computer Science*, pages 589 – 597, 2011. ICCS.