A Method Base for the Situation-Specific Development of Model-Driven Transformation Methods

Marvin Grieger, Masud Fazal-Baqaie, Stefan Sauer
s-lab – Software Quality Lab, Paderborn University
Zukunftsmeile 1, 33102 Paderborn
{mgrieger, mfazal-baqaie, sauer}@s-lab.upb.de

1 Introduction

If a software system is valuable for ongoing business, but resists evolution to ever-changing requirements, it has become legacy. This may be due to the fact that the underlying technological platform has become obsolete. A commonsense solution to this problem is to transform the legacy system into a new environment as part of a migration project. The technical transition is achieved by enacting a *transformation method*. Such a method prescribes the activities to perform, artifacts to generate, roles to involve or tools to use in order to transform the system.

In previous work [1], we motivated that it is critical to use a transformation method that fits to the situation at hand. In general, developing situation-specific methods is supported by method engineering approaches. However, we identified that current approaches do not provide a sufficient degree of *controlled flexibility*. This means, they either do not provide sufficient flexibility to develop a method for the project's situation or they do not sufficiently guide, i.e., control, the development of a method.

To address this problem, we developed a Situational Method Engineering (SME) framework called MEFiSTo [2]. The framework consists of two main constituents, namely a *Method Base* and a corresponding *Method Engineering Process* [3, pp.3-6]. The method base contains predefined building blocks of transformation methods that are systematically assembled by the corresponding method engineering process to form a transformation method. In this paper, describe the content of the method base.

2 Method Base

In MEFiSTo, the method base contains two different types of building blocks of transformation methods, namely *Method Fragments* and *Method Patterns*. Subsequently, we describe both types separately.

Method Fragments The method fragments constitute *atomic building blocks* of transformation methods, i.e., a single *artifact*, *activity*, *role* or *tool*. In Figure 1, the activities and artifacts stored in the method base are illustrated.

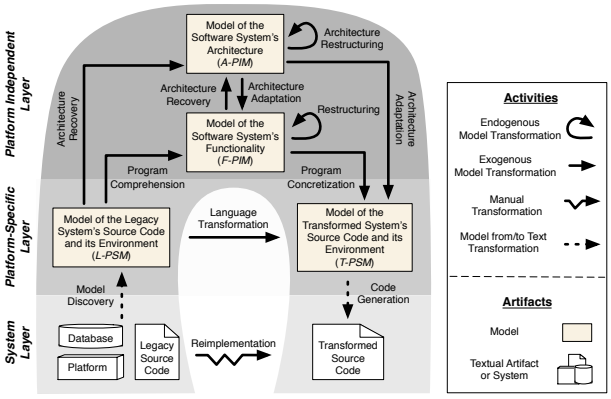

Figure 1: Method fragments stored in the Method Base of the MEFiSTo framework [2]

The fragments have been derived from scratch, based on principles that were defined as part of the Architecture-Driven Modernization (ADM[1]) initiative of the Object Management Group (OMG). ADM aims to transfer model-driven engineering to the domain of software modernization: Models are used on various levels of abstraction and are transformed into each other by model transformations. Conceptually, ADM is related to the Model-Driven Architecture (MDA) by using the same levels of abstraction.

On the system layer, the source code of the legacy system as well as the target system reside. Also, external systems like databases are located there.

Models on the platform-specific layer (PSMs) represent the legacy (*L-PSM*) and the transformed system (*T-PSM*), respectively. The models describe the source code in the form of platform-specific Abstract Syntax Graphs (ASGs) by using metamodels of the corresponding programming languages.

Models on the platform-independent layer (PIMs) are used to represent the functionality (*F-PIM*) and the architecture (*A-PIM*) of the system to transform. The functionality can, for example, be represented in

[1] `http://adm.omg.org`

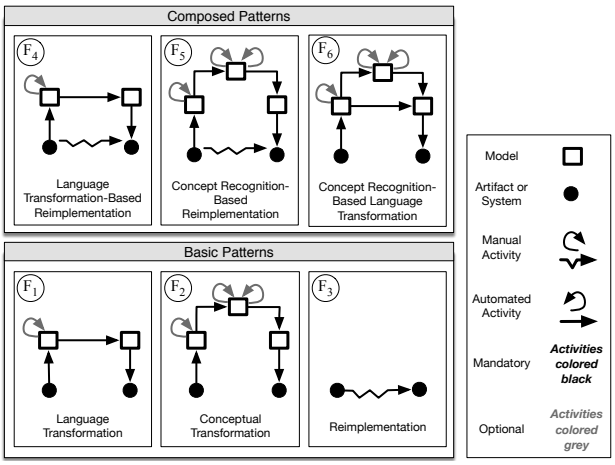

Figure 2: Method Patterns stored in the Method Base of the MEFiSTo framework [2]

the form of a platform-independent ASG by using a metamodel of generic programming language concepts, like loops, conditions, and function calls. However, the F-PIM is not limited to modeling source code, but any information that represents the functionality of the system. Examples being states of the system, structures of user interfaces or dialog flows. The architecture can be represented in the A-PIM by modeling existing components or layers.

Method Patterns The method patterns encode *transformation strategies* by indicating which method fragments to customize. In this way, they provide guidance on how to use the fragments. In total, we observed 14 patterns, six are shown in Figure 2.

We distinguish *Basic* and *Composed Patterns*, whereby basic patterns describe elementary transformation strategies. The *Language* and *Conceptual Transformation* pattern prescribe to automatically convert a legacy system using a model-driven tool chain. Thereby, they differ in the abstraction level to use. When applying the former pattern (F_1), the transformation occurs on the platform-specific layer by defining mappings between the syntactic elements of the programming languages involved. In contrast, when applying the latter pattern (F_2), an intermediate representation of the functionality to transform on the platform-independent layer is used. The *Reimplementation* pattern (F_3) prescribes to manually reimplement functionality by software developers. Composed patterns result when combining one or more basic patterns.

Each method pattern has its own characteristics that determine its suitability in a given situation. For example, we assume that the suitability of the language transformation pattern is mainly determined by the complexity of the equally named activity, i.e., the model transformation between the L-PSM and T-PSM. This activity becomes complex, if the functionality to transform is realized *significantly different* in both environments. In this instance, three different concerns need to be addressed by the model transformation: First, the L-PSM needs to be *interpreted* (I) to identify the functionality to transform. Then, the functionality needs to be *restructured* (II) before being *concretized* (III) in the target environment. In this situation, the conceptual transformation pattern can be a better fit. It reduces the complexity of the transformation by separating these concerns, i.e., it specifies to use a separate activity for each concern.

3 Preliminary Results

We evaluated the MEFiSTo framework in an industrial context, in which we transformed two legacy systems into new environments [2]. Thereby, we successfully developed and enacted situation-specific transformation methods, using the proposed method base.

Based on our experience with the framework, we conclude that, on the one hand, the method fragments provide a high degree of flexibility in the development of a transformation method. They can be used to express the transformation strategy that fits best to the situation at hand. On the other hand, the method patterns provide useful guidance in the development. They can be seen as a terminology, i.e., vocabulary, to discuss one of the most important concerns of a transformation method, i.e., how to transform some functionality. They provide a sufficient degree of abstraction to discuss this concern with different participants of the migration project.

Acknowledgements This work is supported by the Deutsche Forschungsgemeinschaft under grants EB 119/11-1 and EN 184/6-1

References

[1] M. Grieger and M. Fazal-Baqaie. Towards a framework for the modular construction of situation-specific software transformation methods. *Softwaretechnik-Trends*, 35(2):41–42, 2015.

[2] M. Grieger, M. Fazal-Baqaie, G. Engels, and M. Klenke. Concept-based engineering of situation-specific migration methods (to appear). In *Proceedings of the 15th International Conference on Software Reuse (ICSR)*. Springer, 2016.

[3] B. Henderson-Sellers, J. Ralyté, P. J. Ågerfalk, and M. Rossi. *Situational Method Engineering*. Springer, Berlin, Heidelberg, 2014.