Building Transformation Networks for Consistent Evolution of Interrelated Models

Heiko Klare

Context When developing a software system, developers and further stakeholders employ multiple languages or, in general, tools to describe different concerns. Code is often the central artifact, which is, however, implicitly or explicitly complemented by specifications of the architecture, deployment, requirements and others. In addition to a programming language, further languages are used to specify these artifacts, such as the UML for object-oriented design or architecture models, the OpenAPI standard for interface definitions, or Docker for deployment specifications. To achieve a functional software system, these artifacts must depict a non-contradicting specification of the system. For example, service interfaces must be represented in all these artifacts uniformly. We say that the artifacts have to be *consistent*.

Problem In model-driven development, such artifacts are denoted as *models* and constitute central units of the development process, from which also parts of the program code can be derived. This is, for example, already applied in automotive software development [5], but also applies to general software development even if artifacts are not explicitly denoted as models. A common means to preserve consistency between models are transformations [2], which adapt the other models after one of them was changed [1]. Existing research is restricted to *bidirectional* transformations that preserve consistency between pairs of models [10] or to project-specific combinations of transformations to preserve consistency of multiple models [4], as also pointed out in a recent Dagstuhl seminar [6]. Actual software systems are, however, usually described with more than two models, and the languages used to specify them vary between projects. From a software engineer's perspective, reusability of solutions for recurring problems is crucial. This also applies to transformations [8], which, once developed for two specific languages such as UML class diagrams and Java code, may be reused across all projects using the same languages. Even without targeting reuse, supporting independent development of transformations is beneficial to separate concerns, because each domain expert who specifies a transformation may only know about consistency between some of the languages but not between all of them [7]. A systematic engineering process that enables the independent development of transformations and their modular reuse in different contexts is, however, not yet supported.

Gap In this thesis, we research how developers can combine multiple transformations to a network that is able to execute these transformations in an order such that all resulting models are consistent. Existing work puts a rather mathematical view on the problem, i.e., it considers mathematical requirements for the decomposition of consistency between multiple models to a pairwise notion [10] and assumes that transformations can be aligned with each other to interoperate properly [9]. We, in contrast, consider the problem from a software engineer's perspective and make the central assumption that each transformation between two languages is developed independently and that the transformations are not and cannot be aligned with each other, in order to support independent development and reuse. We base mathematically founded as well as empirical considerations from case studies on these assumptions. Our contributions are separated into those concerning the correctness and those concerning the optimization of quality properties of such a combination of transformations to a network.

Contributions We first derive and precisely define an appropriate notion of *correctness* for transformation networks. It induces three specific requirements, which are a *synchronization* property of the single transformations, a *compatibility* property of a network of transformations, and finding an appropriate *orchestration*, i.e., an execution order of the transformations. We propose a construction approach for transformations to fulfill the synchronization property with existing transformation specification languages on a formally proven property. This allows developers to specify transformations that properly interoperate with others without knowing these other transformations beforehand, thus supporting independent development and reuse. For this approach, we show completeness and appropriateness with a case-study-based empirical evaluation in the domain of component-based software engineering. We formally define compatibility of transformations, for which we propose a formal analysis, which is proven correct, and derive a practical analysis, whose applicability we demonstrate with case studies. This supports developers in detecting contradiction between transformations, which can, for example, occur when different developers have conflicting notions of consistency such that the developed transformations cannot preserve them at the same time. Finally, we define the orchestra-

tion problem of finding an execution order that delivers consistent models whenever such an orchestration exists. We prove undecidability of that problem and discuss that restrictions to achieve its decidability will likely limit practical applicability. For that reason, we propose an algorithm that conservatively approaches the problem. It guarantees to deliver an orchestration under specific, well-defined conditions and otherwise indicates an error. We prove correctness of the algorithm and a property that supports finding the cause whenever the algorithm fails. This raises awareness of developers about the inability of any orchestration algorithm to always succeed and also gives them an algorithm at hand that systematically supports them in dealing with failures. Additionally, we categorize errors that can occur if a transformation network does not fulfill the defined correctness notion, from which we derive by means of the mentioned case studies that most potential errors when executing transformation networks can be avoided already by construction with the approaches that we propose in this thesis.

Our investigation of quality of transformation networks is based on a classification of relevant properties, such as maintainability and reusability, and of the effects of different network topologies on them. It reveals that especially correctness and reusability are contradicting properties, thus the selection of a network topology induces a trade-off between these properties. We derive a construction approach for transformation networks that mitigates the necessary trade-off decision and, under specific assumptions, guarantees correctness by construction. We support the development process for this approach with a specification language. While trade-off mitigation is given by construction of the approach, we show achievability of the assumptions and benefits of the proposed language in an empirical evaluation using the case study from component-based software engineering. This makes developers aware of necessary trade-offs between quality properties but also provides them a constructive approach to mitigate them.

Benefits These contributions support researchers as well as transformation developers and users in analyzing and constructing networks of transformations. They depict systematic knowledge about correctness and further quality properties of transformation networks for researchers and transformation developers. In particular, they show precisely which parts of these properties can be achieved by construction, which can be validated by analysis, and which errors must inevitably be expected during execution. Along with these insights, we provide practically applicable approaches for the construction, analysis and execution of correct and modularly reusable transformation networks, from which developers and users both benefit.

Limitations and Future Work While the contributions systematically support the specification of transformation networks to keep multiple artifacts of a software development process consistent, its assumptions raise potential limitations to be investigated in future work. First, the contributions assume models conforming to an object-oriented description according to the *Meta-Object Facility* [3]. While this is sensible for software design, it is unclear whether this is also appropriate for other types of models, e.g., describing describing behavior. In addition, we assume transformations to be a suitable means to preserve consistency between software engineering artifacts. They have not yet been applied to general software engineering in a large scale but in specific domains such as automotive software engineering. In particular, the benefit of reduced development effort and errors through preserving consistency has to amortize the effort for specifying the transformations, which is currently only a sensible expectation but has to be validated empirically in general software engineering.

Outlook Although the specific algorithms and construction approaches we researched may become obsolete or revised at some point in time, the foundational insights regarding decidability of the orchestration problem, regarding the necessity for synchronization and the required properties of transformations to achieve it, as well as regarding the inevitable trade-offs between quality properties are inherent to the problem and will thus remain relevant if discussing solutions to the problem under the same assumptions.

References

[1] P. Stevens. "Bidirectional model transformations in QVT: semantic issues and open questions". In: *Softw Syst Model* 9.1 (2010), pp. 7–20.

[2] A. Kusel et al. "A Survey on Incremental Model Transformation Approaches". In: *Workshop on Models and Evolution*. CEUR-WS, 2013, pp. 4–13.

[3] Object Management Group (OMG). *Meta Object Facility (MOF) Core Specification*. 2016.

[4] Z. Diskin, H. König, and M. Lawford. "Multiple Model Synchronization with Multiary Delta Lenses". In: *21st International Conference on Fundamental Approaches to Software Engineering*. Springer, 2018, pp. 21–37.

[5] H. Guissouma et al. "An Empirical Study on the Current and Future Challenges of Automotive Software Release and Configuration Management". In: *44th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 2018, pp. 298–305.

[6] A. Cleve et al. "Multidirectional Transformations and Synchronisations (Dagstuhl Seminar 18491)". In: *Dagstuhl Reports* 8.12 (2019), pp. 1–48.

[7] H. Klare et al. "A Categorization of Interoperability Issues in Networks of Transformations". In: *Journal of Object Technology* 18.3 (2019), 4:1–20.

[8] J.-M. Bruel et al. "Comparing and classifying model transformation reuse approaches across metamodels". In: *Softw Syst Model* 19.2 (2020), pp. 441–465.

[9] P. Stevens. "Connecting software build with maintaining consistency between models: towards sound, optimal, and flexible building from megamodels". In: *Softw Syst Model* 19.4 (2020), pp. 935–958.

[10] P. Stevens. "Maintaining consistency in networks of models: bidirectional transformations in the large". In: *Softw Syst Model* 19.1 (2020), pp. 39–65.