Towards a Correspondence Model for the Reuse of Software in Multiple Domains

Sandro Koch, Frederik Reiche
{sandro.koch, frederik.reiche}@kit.edu
Karlsruhe Institute of Technology

Abstract

Modern software systems are often reused in multiple domains and for various purposes. To allow this ambiguity, the connection and transformation between the systems has to be described. In this paper we present an idea of a Domain-Specific Language (DSL) that enables to describe the connection and the transformation distinctly. The separation of the description of which element is connected to which and the transformation description eases future reuse and improves the maintainability.

1 Introduction

The development of modern systems may require the reuse of existing elements that are not explicitly designed for the intended use case. These elements may be developed beforehand or are purchased of a third party. Independently designed interfaces are a common way to separate software components without relying on the concrete implementation [1]. But when the interfaces do not match, further adjustments are necessary. The adapter pattern [2] for instance, allows the integration of a component into a different context, despite the mismatch of the interfaces. The need for strategies to cope with the problem of relating elements in different contexts can also be found in other domains of software development. For instance, Koch et al. [3] present a model-based approach to describe the coupling of simulations. Conversions between simulations are realized by descriptions of required and provided data and their transformation. In their approach to provide a better end-to-end security analysis, Kang et al. [4] bridge different vocabularies in the different views on the system by introducing mappings between model elements. Seifermann et al. [5] discuss follow-up issues in the results of their data-flow analysis of software architectures. A mapping and filtering of the results to the architectural model elements is proposed. In the field of change propagation analysis in Cyber-physical System (CPS) Heinrich et al. [6] use a correspondence model to map mechanical and electrical components of a CPS to their corresponding variables. Busch et al. [7] utilizes the results of the mapping in a CPS to allow a change propagation in the control software. Heinrich et al. [8] apply a Runtime Architecture Correspondence Model to relate monitoring data to the elements of the architectural model under analysis. In this paper we introduce a DSL that allows to describe the correspondence independently of the domain.

The paper is organized as follows. Section 2 outlines the initial problem statement. Section 3 introduces the correspondence DSL. Section 4 concludes the paper and outlines future work.

2 Problem Statement

For this paper, we will investigate the correspondence between two elements. A correspondence is the mapping of one element to another, with an optional transformation. This transformation used in a correspondence can be unidirectional or bidirectional. In the simple case, one or multiple types must be converted to another type (e.g., source: integer to destination: float). The simple case can be solved with an adapter which transforms the source type into the destination type. However, if concepts or elements from different domains need to be mapped or when there are many elements need to be linked, the complexity of the adapter increases. Thus, we distinguish two problems regarding the reuse and maintainability: The first problem (`P1:` *Implicit Modeling*) occurs, because correspondences are not easily identifiable. Correspondences are an implicit part of the transformations and not explicitly modeled. The second problem (`P2:` *Complexity*) occurs, because the transformations are a result of the underlying domain. A complex model can lead to complex transformations. `P2` is directly dependent on `P1`. For example, in a change-impact analysis which allows to determine the impact of a change of mechanical/electrical elements to corresponding architecture elements in the control software [6] each impact must be described separately. As a result, the reuse and maintainability of an adapter and its transformation is hindered.

3 Language

The goal of the correspondence language is to allow the modeling of a connection between two entities and a transformation if necessary. An entity is defined

```
//Entities
entity Airplane extends SimEntity:
  provides Velocity

entity Jet extends SimEntity:
  requires Speed

//Exchange Types
exchange Velocity:
  int: Magnitude
  vector3D: Direction

exchange Speed:
  int: Magnitude

//Correspondence
correspondence of airplane and jet:
  if input is Airplane then use
   velocity_to_speed

//Rules
rule velocity_to_speed:
  return Airplane.Velocity.Magnitude
```

Listing 1: "Correspondence Language Example"

with the keyword `entity`. An excerpt of the DSL is depicted in Listing 1. It shows the correspondence between two entities `SimEntity` named "Airplane" and "Jet". As the name `SimEntity` implies, the two entities "Airplane" and "Jet" are software systems, more precisely simulations, which need to be coupled. Each entity can define `required` and `provided` fields. In our example, we define two `exchange` types. The first is "Velocity" which has a magnitude and a direction. The second is "Speed" which has a magnitude but no direction. The "Airplane" simulation uses velocity and the "Jet" uses speed. In order to define a correspondence between these two entities, the language provides the keywords `correspondence of` which connects two entities. If the airplane simulation sends data to the jet simulation, the specified transformation rule "velocity_to_speed" needs to be called. The rule is simplified and returns the magnitude of the airplane. In future iterations of the language, it is planned to allow more complex transformations. The idea is to allow the integration of transformation languages like QVT and ATL. The utilization avoids to implement another transformation language and set the focus on the structure of the correlation.

The problem `P1` is tackled by the explicit modeling of each correspondence and the separation of the transformation rules and their depending in- and output. The problem `P2` is tackled by the simplicity of the DSL and the possibility of splitting code in multiple files. The solution of these problems results in a structuring of the correlation, thus a better maintainability and the possibility of reusing transformations for different correlations can be achieved.

4 Future Work

For the future it is planned to provide and integrate a set of predefined transformations in the language. The idea is to include name mappings (e.g., input: speed, output: Speed) and type conversions (e.g., hour to seconds) as language features. Here template solutions for classes of transformation will be researched to provide support in the application of the language. Also, further research on how to improve the transformation description while keeping it also light-weight in the language is possible. The generation of code from the correspondences and transformations is another topic that will be realized. Furthermore, it will be researched how correspondences between source-code elements and model elements can be realized within the language.

Acknowledgement

This work was supported by the Ministry of Science, Research and the Arts Baden-Württemberg in the funding line Research Seed Capital (RiSC) and by the German Federal Ministry of Education and Research within the project "Security of Connected Infrastructures" in the Competence Center for Applied Security Technology (KASTEL).

References

[1] I. Sommerville. *Software Engineering, Global Edition.* Vol. 10. Pearson Education Limited, 2016. 816 pp.

[2] E. Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software.* Pearson Education, 1994. 457 pp.

[3] S. Koch, F. Reiche, and R. Heinrich. "Towards a Metamodel for Modular Simulation Environments". In: *Models and Evolution Workshop, MODELS'18.* CEUR-WS, Oct. 2018.

[4] E. Kang, A. Milicevic, and D. Jackson. "Multi-representational security analysis". In: *FSE.* ACM Press, 2016, pp. 181–192.

[5] S. Seifermann, R. Heinrich, and R. Reussner. "Data-Driven Software Architecture for Analyzing Confidentiality". In: *ICSA'2019.* IEEE, 2019.

[6] R. Heinrich et al. "Architecture-based change impact analysis in cross-disciplinary automated production systems". In: *Journal of Systems and Software* 146 (2018), pp. 167–185.

[7] K. Busch et al. "A Model-Based Approach to Calculate Maintainability Task Lists of PLC Programs for Factory Automation". In: *IECON'18.* IEEE, Oct. 2018, pp. 2949–2954.

[8] R. Heinrich et al. "Software Architecture for Big Data and the Cloud". In: Elsevier, 2017. Chap. An Architectural Model-Based Approach to Quality-aware DevOps in Cloud Applications.