A Co-evolution Approach for Source Code and Component-based Architecture Models*

Michael Langhammer
Karlsruhe Institute of Technology
Karlsruhe, Germany
michael.langhammer@kit.edu

Klaus Krogmann
FZI Research Center for Information Technology
Karlsruhe, Germany
krogmann@fzi.de

Abstract

During the lifecycle of a software system, the software needs to evolve, e.g, through new features or necessary platform adaptions. If architecture and source code are not kept consistent during this software evolution, well-known problems, such as architecture drift and architecture erosion, can occur.

To solve these problems, existing approaches usually focus on the consistency between class diagrams and code, or use approaches where the architecture model can completely be generated from the code.

In this paper, we present a fully integrated co-evolution approach for component-based architecture and source code based on Vitruvius. We also present initial, extendable mapping rules from component-based architecture to source code.

1 Introduction

Architecture drift and architecture erosion are two well known problems, which can occur during software evolution [7]. They can occur e.g., if code evolves independently from the system's architecture.

In this paper, we present a co-evolution approach that helps software architects and developers to prevent architecture drift between source code and component-based software architecture. This approach is based on a view-centric engineering approach called Vitruvius [5, 6]. Vitruvius (see Figure 1) can be used to keep heterogeneous models consistent during the development of a system. It is based on the idea of having all information that belong to a software system stored within a SUM (Single Underlying Model) [2]. The access to this SUM is only possibles via well-defined views. While a SUM eases accessing a single underlying information source, it is hardly applicable in practice since it requires a single world model. To reuse existing meta-models and models within Vitruvius we have introduced the idea of a so called VSUM (Virtual Single Underlying Model) [5], which orchestrates all individual used models without extending or changing models or meta-models. Within Vitruvius, mapping rules describe the overlap between heterogeneous models of the VSUM. This can be done either by using the MIR (Mapping Invariant Response) language, which we are currently developing, or using Xtend[1]. The MIR or Xtend rules transform changes among models. Therefore, Vitruvius monitors all views respectively editors acquire atomic changes. These atomic changes are used as input for the consistency preserving transformations, which translate changes element by element.

2 Co-evolution approach

The current focus of our work is the application of Vitruvius to component-based software architecture and code (see Figure 1). Vitruvius solely operates on models. As a component-based architecture model, we use the PCM (Palladio Component Model) [3]. The PCM offers users the creation of a component-based architecture in terms of components and interfaces.To get a model representation of the source code, we use JaMoPP (Java Model Parser and Printer) [4], which extracts an EMF-based (Eclipse Modelling Framework) representation of Java code.

To use Vitruvius we have implemented a monitor for the Eclipse Java code editor and for EMF-based models [6]. We also defined the following initial mapping rules from component-based architecture models to Java source code: A PCM *repository* is represented by three packages: one *main package*, one *datatype package*, which contains all datatypes, and one *contracts package*, which contains all *interfaces*. A *BasicComponent* is mapped to a package within the main package and a component-class. This class *implements* all interfaces the component provides. To realize *RequiredRoles* we use the dependency injection pattern: the class has a *field* and a *constructor parameter* with the type of the required interface. Using this mapping rules, our current prototype is able to round-trip architecture and code. Our extensible mapping rules will be complemented by a mapping to Eclipse Plugins and a dependency injection frameworks in our ongoing work. It is also possible to create project-specific mapping rules for individual projects. Therefore, it is possible to use the MIR language from

*Acknowledgment: This work funded by the German Research Foundation in the Priority Programme SPP1593

[1] http://eclipse.org/xtend/

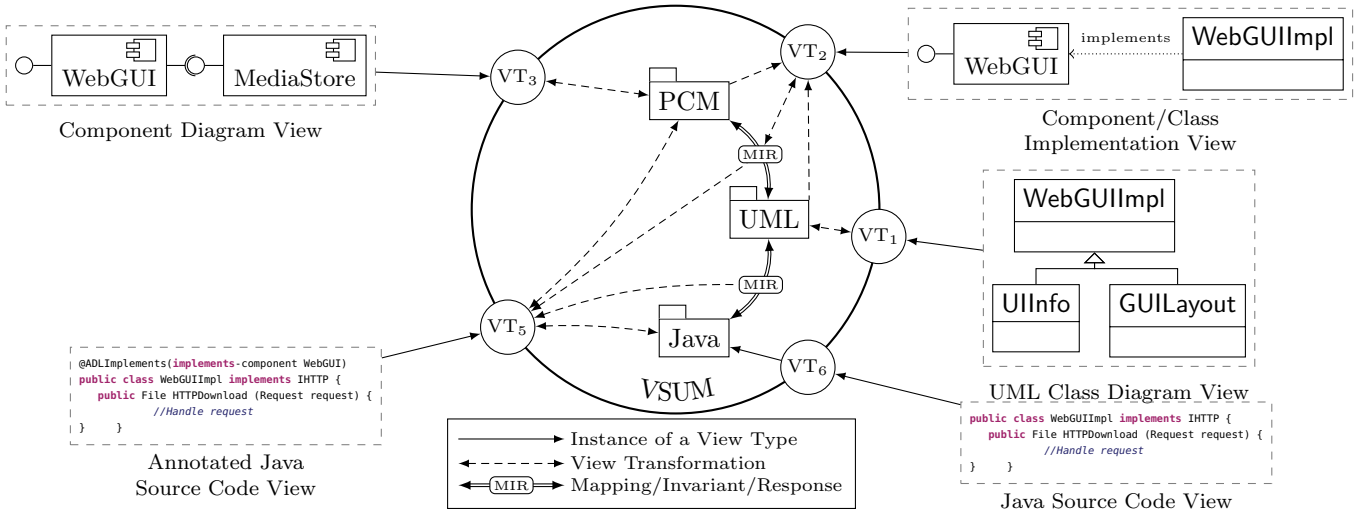

Figure 1: Application of VITRUVIUS to component-based software engineering with multiple views. For our first prototype we only use PCM and JaMoPP as meta-models within the VSUM (Virtual Single Underlying Model) and the standard component views for PCM and the source code view for JaMoPP.

VITRUVIUS or Xtend to extend one of the pre-defined mappings or to create new *mapping* rules. MIR *invariants* declare, for example, constraints on models, which arise from consistency requirements. MIR *responses* define actions to circumvent violations of MIR invariants by creating and deleting model elements. Another concept of our approach are the user interactions. These are used if developers or architects perform an ambiguous change. An ambiguous change is a change that cannot be transformed automatically to architecture respectively code. For instance, if developers add a new interface in Java it is unclear whether it should be reflected on the architecture level as well or if it is just a technical interface. To clarify the intend we currently use dialogs, where developers have to choose between the different options.

3 Related work

Existing approaches, e.g., IBM Rational Rhapsody[2], support round-trip engineering but rely on source code as the single information source and generate other representations, such as class diagrams from it. Other approaches, e.g., UMLLab[3], ensure consistency between class diagrams and code, but not between component diagrams and source code. ArchJava [1] includes architectural constructs (e.g., ports) into the source code itself while our approach is not invasive.

4 Conclusion

In this paper, we presented our co-evolution approach for component-based software architecture and source code, which is based on the VITRUVIUS approach. Our co-evolution approach helps software developers and architects evolving their software system by keeping the architecture and the source code consistent during the evolution of a software system.

[2]http://www-03.ibm.com/software/products/de/ratirhapfami
[3]http://www.uml-lab.com/

References

[1] J. Aldrich, C. Chambers, and D. Notkin. “ArchJava: connecting software architecture to implementation.” In: *Proceedings ICSE 2002*. IEEE. 2002, pp. 187–197.

[2] C. Atkinson, D. Stoll, and P. Bostan. “Orthographic Software Modeling: A Practical Approach to View-Based Development”. In: *Evaluation of Novel Approaches to Software Engineering*. Springer, 2010.

[3] S. Becker, H. Koziolek, and R. Reussner. “The Palladio component model for model-driven performance prediction”. In: *Journal of Systems and Software* 82.1 (2009), pp. 3–22.

[4] F. Heidenreich et al. “Closing the gap between modelling and java”. In: *Software Language Engineering*. Springer, 2010, pp. 374–383.

[5] M. E. Kramer, E. Burger, and M. Langhammer. “View-centric engineering with synchronized heterogeneous models.” In: *Proceedings of the 1st Workshop on VAO*. ACM. 2013.

[6] M. E. Kramer et al. “Change-Driven Consistency for Component Code, Architectural Models, and Contracts.” In: *Proceedings of the 18th International ACM Sigsoft Symposium on CBSE*. accepted, to appear. ACM, 2015.

[7] D. Perry and A. Wolf. “Foundations for the study of software architecture”. In: *ACM SIGSOFT Software Engineering Notes* (1992).