ReflexML: UML-Based Architecture-to-Code Traceability and Consistency Checking

Josef Adersberger\textsuperscript{1} and Michael Philippsen\textsuperscript{2}

\textsuperscript{1} QAware GmbH, Aschauer Str. 32, 81549 Munich, Germany
josef.adersberger@qaware.de
\textsuperscript{2} University of Erlangen-Nuremberg, Computer Science Department, Programming Systems Group, Martensstr. 3, 91058 Erlangen, Germany
philippsen@cs.fau.de

Abstract. The decay of software architecture - the divergent evolution of architecture models and the derived code - is one of the reasons for a decreasing maintainability of software systems. Several approaches for architecture-to-code consistency checking exist that stop the decay by detecting a divergence after evolution steps of either the architecture or the corresponding code. Known approaches have two main insufficiencies. First, the effort to derive and maintain the consistency checks is higher than necessary or they cannot be applied a posteriori. Second, they are not well integrated into UML-based model driven engineering. In the paper we present ReflexML: A UML-embedded mapping of architecture models to code plus a rich set of predefined consistency checks based on that mapping. The mapping is described with a UML profile that allows to attach AOP type patterns to an UML component model to define its reflexion on code elements. This abolishes the two insufficiencies of current approaches. We apply ReflexML to an industry project to demonstrate its effectiveness and its capability of a seamless integration into a pre-existing UML architecture model.

Keywords: traceability, reflexion model, architecture consistency, UML, AOP.

1 Introduction

Maintainability of software is mainly driven by its architecture. Software architecture has a classic divide-and-conquer core: All concerns of a software are split into coherent parts (components) which are then loosely coupled via clearly defined interfaces. Most modern development methodologies require to develop a software architecture and derive the implementation structure from there. But then in many real-world projects the architecture and the implementation structure diverge while the software evolves. This leads to a decreasing maintainability called architecture erosion\textsuperscript{15} that is a typical effect of software aging\textsuperscript{13}. Architecture erosion can be prevented by enforcing so-called architecture-to-code consistency, i.e., if code-level dependencies comply with the dependencies and component semantics defined on the architecture level.
There are two types of such inconsistencies [9] or architecture violations [15], namely divergence (a dependency between two code elements is not allowed according to the architecture model) and absence (a dependency in the architecture model is not represented in the code). To stop architecture erosion, methods are needed that perform architecture-to-code consistency checks after each evolution step of either the architecture model or the code.

In general, any approach to check architecture-to-code consistency has to address the following two topics: First, architecture-to-code traceability is required to map architecture elements to code elements. This mapping is also known as reflexion model [9]. Without being able to identify the associated code elements of an architecture element, no architecture compliance check can be applied. Second, architecture compliance checks are required to reveal the consistency between architecture and code. Such checks detect dependencies on code-level that are disallowed according to the architecture model.

We think that any approach should implement the following five requirements to be applicable in all scales of real-world projects. These requirements also solve the two main insufficiencies of current approaches: The high effort to define and maintain both the required mapping and the compliance checks as well as the lacking integration into model-driven engineering.

1. Single source: The native architecture models should be used as input for the approach to avoid duplication of architecture information. Otherwise maintaining both the basic architecture models and the consistency checking models would be costly and error-prone.
2. Expressive mapping: Defining every relationship by enumerating all mapped code elements is infeasible in large projects. Instead we need an expressive query-like way to describe the mapping of architecture elements to code elements to reduce the effort to create and maintain the mapping.
3. Stable mapping: The mapping should be stable with the evolution of the architecture model or the code. If the architecture is refactored or code elements are created, moved, or deleted the mapping should either stay valid or needed modification should be simple to identify.
4. Semantically rich architecture model: With semantically rich constructs like components, interfaces, and hierarchical compositions a rich set of pre-defined architecture compliance checks can be expressed. Semantically poorer models require to define a large set of explicit rules to ensure architecture compliance. E.g., the semantically poorer boxes-and-lines model only allows to define valid dependencies (the lines) between sets of classes (the boxes).
5. A priori and a posteriori application: It should be possible to perform compliance checking both from the start of the implementation of an architecture or during/after the implementation.

We present ReflexML that addresses both topics and fulfills all of the above requirements. We limit the scope of ReflexML according to the following two assumptions. First, the architecture model is described with UML component models. Alternative to UML a couple of other architecture description languages