Connecting Architecture and Implementation

Georg Buchgeher¹ and Rainer Weinreich²

¹ Software Competence Center Hagenberg, Austria
giorg.buchgeher@scch.at
² Johannes Kepler University Linz, Austria
rainer.weinreich@jku.at

Abstract. Software architectures are still typically defined and described independently from implementation. To avoid architectural erosion and drift, architectural representation needs to be continuously updated and synchronized with system implementation. Existing approaches for architecture representation like informal architecture documentation, UML diagrams, and Architecture Description Languages (ADLs) provide only limited support for connecting architecture descriptions and implementations. Architecture management tools like Lattix, SonarJ, and Sotoarc and UML-tools tackle this problem by extracting architecture information directly from code. This approach works for low-level architectural abstractions like classes and interfaces in object-oriented systems but fails to support architectural abstractions not found in programming languages. In this paper we present an approach for linking and continuously synchronizing a formalized architecture representation to an implementation. The approach is a synthesis of functionality provided by code-centric architecture management and UML tools and higher-level architecture analysis approaches like ADLs.

1 Introduction

Software architecture is an abstract view of a software system. As such it abstracts from implementation details and data structures [1] and describes important elements, externally visible properties of these elements, and relationships among elements [1]. Different approaches for describing software architecture exist. Informal approaches and UML diagrams are typically used for architecture documentation. Formal approaches like Architecture Description Languages (ADLs) and architecture models [2] are used for automatically analyzing specific system properties.

Keeping an architecture description up to date and ensuring that the prescriptive (as-intended) architecture corresponds to the descriptive (implemented) architecture are still central problems in software development [2][8][4]. Approaches addressing these problems exist. For example, UML-tools may extract architectural abstractions from an implementation and thus provide a view of the currently implemented architecture. Architecture management tools like Lattix [5], SonarJ and Sotoarc [6] additionally support comparison and synchronization of
the intended and the implemented architecture. However, these approaches operate on the concepts at the low abstraction level provided by object-oriented programming languages (classes and interfaces). They fail to support higher-level abstractions like components, systems, and systems of systems. Typically they also work only for homogenous programs and do not inherently support heterogeneous software systems, like service-oriented software architectures. Approaches extracting architecture information from code also typically lack validation capabilities like ADLs. On the other hand, general-purpose ADLs like xADL [7] and ACME [8] provide only limited support for connecting architecture description and code and offer no synchronization support.

In this paper we describe an approach for connecting and synchronizing architecture description and system implementation. The approach is a synthesis of functionality provided by code-centric architecture management and UML tools and higher-level architecture analysis tools like ADLs. It supports both the description of high-level architectural elements like components and systems and of low-level concepts like classes and interfaces. Low-level abstractions are extracted from an implementation similar to reverse engineering and architecture management tools and can be synchronized with a prescribed architecture. High-level abstractions are either defined manually or extracted from a system implementation. In the latter case, the system analyses technology-specific component code, meta-data and configuration files to extract the needed information. Since we use an ADL for architecture representation, the approach also supports architecture validation based on constraints. Heterogeneous systems are supported through different technology bindings.

The remainder of this paper is structured as follows. In Section 2 we describe the basic elements of our approach, which are an ADL for architecture description and a toolset working on this ADL. The toolset provides visual editors for visualizing and manipulating an ADL-based architecture model, functionality for extracting information from an implementation, and functionality for synchronizing prescriptive and descriptive architecture. In Section 3 we provide an example to illustrate our approach. In Section 4 we describe related work in more detail. The paper is concluded in Section 5.

2 Approach

The main elements of our approach are an ADL, called LISA (Language for Integrated Software Architecture), and a toolkit working on LISA-based architecture models, the LISA-Toolkit (see Figure 1). The toolkit is implemented on the basis of the Eclipse platform and provides a number of plug-ins for architecture modelling, visualization, validation, and implementation synchronization. Some of the available editors and views for architecture modelling and visualization are shown in the next section.

The LISA-ADL is an extensible XML-based language for representing structural relationships of heterogeneous software systems. It is implemented by XML-based metamodels for describing different aspects of a software system. We