An Architecture for Toolbox-based Software Architecture Reconstruction Solutions

Ana Dragomir, M. Firdaus Harun, Horst Lichter
Research Group Software Construction
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
{ana.dragomir, firdaus.harun, horst.lichter@swc.rwth-aachen.de}

ABSTRACT

Although up-to-date architecture views outstandingly aid the understanding, meaningful evolution and evaluation of software systems, software architecture reconstruction tools are still not broadly employed in the industry. While this situation might seem contradictory, we have identified - based on our experience with two industry cooperation partners - some important improvement potentials that could increase the usefulness of the current state of the art reconstruction approaches and thus lead to a better industry acceptance. In this paper we present a general architecture for constructing tool-box-based software architecture reconstruction solutions that enable the flexible integration of various analysis plug-ins on a per need basis while addressing the improvement directions identified earlier. Furthermore, we present an instantiation of this architecture that enables the reconstruction of architecture behavior views on more abstraction levels. We conclude the paper with a tool implementation overview of the latter architecture.

1. INTRODUCTION

Up-to-date software architecture descriptions can be of enormous help, when software architecture migration and modernization activities need to be performed. Understanding the dependencies of the various software systems as well as the internals of each system in insolation is crucial and paves the way to more accurate architecture evaluations and goal-oriented evolution. In our previous work [1], we have described the current state of the practice of two of our industry cooperation partners with respect to the need and use of software architecture reconstruction tools to monitor and evaluate complex industrial software landscapes. In both companies, the architecture descriptions are elaborated manually, resulting into considerable, continuous effort invested into ensuring their consistency with the implemented architectures, on the one side, or the emergence of very implementation-close descriptions that lack an abstract view of the described systems or the existence of only early elaborated high-level views that can be hardly associated with the actual implementation, on the other side. None of the two industry partners employ architecture reconstruction tools, although the current state of the art would seem to improve their current situation. We have identified several reasons why this is the case: (1) while some current solutions do address the reconstruction of heterogeneous systems [3], they are not actively used and their setup can be rather complicated; (2) metrics that guide the architects to understand where to start the architecture improvement from are often missing; (3) the evolution of the reconstructed architectures is not properly supported, for example by enabling the architects to define evolution variants and identify the most convenient ones; (4) reconstruction tools often use their specific architecture description terminology (e.g., Sonargraph-Architect [4] allows only the definition of layers, layer groups, vertical slices, vertical slices groups and subsystems) which is often rejected by the architects, as it does not correspond to their understanding of their systems.

In order to address these issues we propose an **a**rchitecture **m**onitoring and **a**nalysis **i**nfrastructure (ARAMIS) that contains an extendable tool-box to which architecture monitoring and analysis plug-ins can be added on a per need basis.

2. ARAMIS - OVERVIEW AND EXAMPLE

In the following we first present the general architecture of ARAMIS and then we give an instantiation example that focuses on the architecture visualization and the analysis of architecture violations during run-time

2.1 ARAMIS - General Architecture

Using ARAMIS (Figure 1) architects should be able to extract architectural information from various, possibly heterogeneous systems in order to facilitate the recreation of more complex architecture landscapes. To address this, it should allow the easy addition and integration of technology-specific *architecture information collectors*. Consequently, if the extracted information is too heterogeneous (e.g., information extracted from procedural code vs. information extracted from object oriented code) an intermediate processing step that normalizes it to a common intermediate representation might be required. The collected information can then be stored in an *architecture repository* and/or can be used directly for further analyses. Based on the extracted information or on the intermediate representation, the architects should be able to define the systems' higher level abstractions, using exactly the same terminology that they are using in their day-to-day practice. This should be possible, even if the architects work with different abstractions in different projects (e.g., Cobol systems are organized in subsystems, Java systems are organized in components and the components are further organized in layers, etc.). The definition of the various architecture description languages that comprise these abstractions, as well as their relation to each other will thus constitute a so-called *architecture modeling language family*. Once the extracted information is mapped

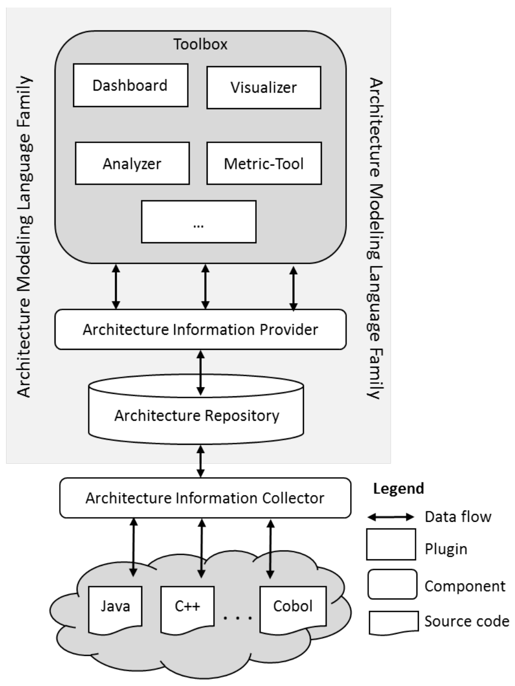
Toolbox
Dashboard
Visualizer
Analyzer
Metric-Tool
...
Architecture Modeling Language Family
Architecture Modeling Language Family
Architecture Information Provider
Architecture Repository
Architecture Information Collector
Java
C++
. . .
Cobol
Legend
Data flow
Plugin
Component
Source code

Figure 1: ARAMIS - General Architecture

on the abstractions chosen by the architects, various plug-ins can be employed to further facilitate the analysis of the results. E.g., based on the reconstructed views, metric plug-ins can be added to compute structural or behavioral metrics and/or to allow the definition and assessment of architecture evolution variants. To gain access to the architecture data necessary for their analyses, the *plug-ins* will register themselves to an *architecture information provider* which will consequently forward them relevant data whenever this is extracted and/or persisted in the architecture repository by any of the employed architecture information collectors.

2.2 ARAMIS - Instantiation Example

We have developed a first instantiation of ARAMIS, that aims to analyze the architecture of software systems during run-time (called ARAMIS-RT). The purpose of ARAMIS-RT is to generate real-time visualizations of data monitored during the run-time. The data should be visualizable on multiple abstraction levels (communication between layers, communication between components, etc.). In order to define these levels of abstraction, we are currently providing the architects with a very simple architecture modeling language that they can use to define their architecture model of interest. As shown in Figure 2, the meta-model simply depicts *architectural units* that can be further contained in other *architectural units*. An architecture unit can have a role (e.g. “layer”, “component”, “filter”, etc.) and can be associated with various software units, as an abstraction thereof. The software units are easily mappable on the data extracted at run-time and represent the entities (in this case - the class instances) between which the intercepted communication occurs. Furthermore, the architect can define architecture communication rules for the defined architecture levels, and then check if these are obeyed or violated during the execution of the system. The data is transmitted

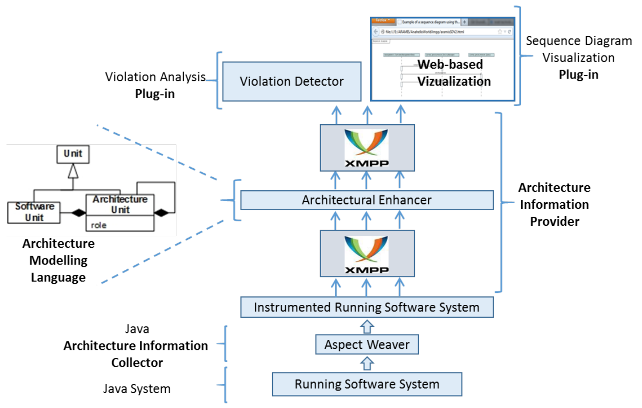
Sequence Diagram Visualization Plug-in
Web-based Vizualization
Violation Analysis Plug-in
Violation Detector
XMPP
Architectural Enhancer
Architecture Information Provider
Unit
Software Unit
Architecture Unit
role
Architecture Modelling Language
XMPP
Instrumented Running Software System
Java Architecture Information Collector
Aspect Weaver
Java System
Running Software System

Figure 2: ARAMIS-RT - An ARAMIS Implementation

in real-time via the XMPP communication protocol. Thus, it is even possible for the architects to trigger the execution using the user interface of the application and then observe almost instantly the way the various architectural elements interacted and if violations occurred. In this case, the XMPP server plays the role of the architecture information provider, as it uses a publish/subscribe mechanism to redirect the data collected during run-time to the two registered plug-ins. Because we only interested in real-time analyses, we did not include an architecture repository to provide data storage for post-mortem investigations. This will be however part of our future work. While ARAMIS-RT is still under development, the results of the first evaluations that we have performed look very promising [2].

3. CONCLUSION

In this paper, we have proposed a general architecture for creating toolbox-based software reconstruction solutions that aim towards improving the current state of the art and achieving a better industry acceptance. We have then presented an instantiation of this architecture that aims at the real-time monitoring of Java systems on more abstraction levels and at the analysis of violations occurring during system execution. In the future, we will further evolve and evaluate ARAMIS-RT presented in this paper and assess its industrial acceptance.

4. REFERENCES

[1] A. Dragomir, M. F. Harun, and H. Lichter. On bridging the gap between practice and vision for software architecture reconstruction and evolution – a tool perspective. SAEroCon Workshop 2014, Sydney, Australia, April 8, 2014.

[2] A. Dragomir and H. Lichter. Run-time monitoring and real-time visualization of software architectures. In *Proceedings of the 20th Asia-Pacific Software Engineering Conference*, December 2013.

[3] J. Ebert, B. Kullbach, V. Riediger, and A. Winter. Gupro: Generic understanding of programs – an overview. In *Electronic Notes in Theoretical Computer Science*, 2002.

[4] Sonargraph-Architect. `https://www.hello2morrow.com/products/sonargraph/architect`, 2013.