Teaching Software Architecture Concepts with HUSACCT

Tool Demo

Christian Köppe
HAN University of Applied Sciences, Arnhem/Nijmegen, the Netherlands
christian.koppe@han.nl

Leo Pruijt
HU University of Applied Sciences, Utrecht, the Netherlands
leo.pruijt@hu.nl

Abstract

Teaching software architecture (SA) in a bachelor computer science curriculum can be challenging, as the concepts are on a high abstraction level and not easy to grasp for students. Good techniques and tools that help with addressing the challenging SA aspects in a didactically responsible way are needed.

In this tool demo we show how we used the software architecture compliance checking tool HUSACCT for addressing various concepts of SA in our courses on software architecture. The students were introduced to architectural reconstruction and architecture compliance checking, which helped them to gain important insights in aspects such as the relation between architectural models and code and the specification of dependency relations between architecture elements as concrete rules.

Categories and Subject Descriptors K.3.2 [*Computers and Education*]: Computer and Information Science Education —Computer science education

General Terms Software Architecture, Education

Keywords Architecture Reconstruction, Architecture Conformance Checking

1. Introduction

Software architecture is a complex knowledge area, where many concepts are on a high abstraction level. Addressing all these concepts in a way that the students can easily grasp them is challenging.

In our experience, many students perform well at the lower level concepts of programming and small-scale designing. The more abstract the concepts become, the more difficult it is for them to grasp these concepts and connect their programming knowledge to these high-level architectural concepts. We observed that students often develop high-level architectural models such as component diagrams or layered architectures, but somehow don't see them directly related to the system to be built based on these models. In the implementation of such systems, the architecture is often not completely realized as intended. Reasons are many-fold: low-level issues the students solve in the most straight-forward way, hereby violating the architectural decisions made before; insufficient knowledge about how the models translate to concrete implementations; or the perception that architecture needs to be done in larger projects, but is not relevant or valuable when it comes to the concrete realization.

Finding ways to help students with better understanding architectural concepts and supporting them with connecting these concepts to their prior knowledge on programming and software design would improve software engineering education. We believe that using a compliance checking tool such as HUSACCT offers possibilities to address these issues in a holistic way. The need for defining the architecture in such a tool including the associated rules that are an essential part of it require the students to think further than just making box-and-arrow diagrams. The students hereby have to ensure that they make correct use of the different architectural elements and rule types in order to define a correct representation of the architecture. Furthermore, the necessary correct mapping of these architectural elements to source code artifacts makes this connection explicit and shows the students that changing one of them is likely of direct influence on the other one.

In the next section we shortly describe the HUSACCT tool. We then explain how the tool is used in our software architecture courses. We conclude with an elaboration on the educational value of using HUSACCT.

2. The HUSACCT tool

HUSACCT (HU Software Architecture Compliance Checking Tool) was initially developed by a group of 25 students during an advanced software architecture course at the HU University of Applied Sciences and has been improved since. It supports compliance checking of semantically rich modular architectures (SRMA): "expressive modular architectures, composed of different types of modules, which are constrained by different types of rules; explicitly defined rules, but also rules inherent to the module types" [5]. This extensive SRMA support offers good opportunities for applying it also for educational purposes.

The following summary is quoted from [6], where an extensive description of the tool is given:

> "HUSACCT is a tool that provides support to analyze implemented architectures, define intended architectures, and execute conformance checks. Browsers, diagrams and reports are available to study the decomposition style, uses style, generalization style and layered style [3] of intended architectures and implemented architectures. HUSACCT is free-to-use and open source. It has been developed in Java and analyzes Java and C# source

presented at SPLASH-E 2015, October 26, 2015, Pittsburgh, PA, USA.

code. The executable and source code are downloadable at `http://husacct.github.io/HUSACCT/`. An introduction video and documentation are accessible at the same site."

3. HUSACCT in a software architecture course

Both authors use HUSACCT in courses on software architecture at their universities. The courses are both part of bachelor curricula in Computer Science with a strong focus on Software Engineering. The courses at the HU University of Applied Sciences are at the 2nd year (introductory level) and 3rd year (advanced level). The course at HAN University of Applied Sciences is at the 3rd year.

In the courses, two of the topics covered that make use of the HUSACCT tool are *architecture reconstruction* and *architecture compliance checking*. The lectures on these topics consist of theoretical explanation and hands-on exercises. The results of these exercises are immediately discussed in class. Additionally, take-home assignments are given which implicitly address other important aspects too (as described in section 4). Example systems for the practical parts were HUSACCT itself (version 1.0) for the take-home assignments and an open source card game for the in-class exercises. Both are implemented in Java, which the students are familiar with, and had the source code available.

The following sections describe the take-home assignments in more detail.

3.1 Assignment 1: Architecture Reconstruction

This assignment is intended for making the students familiar with HUSACCT and to introduce *architecture reconstruction*, a process of reverse engineering that allows an analyst "to build, maintain, and understand a representation of an existing architecture" [2].

The students get the source code of HUSACCT itself (version 1.0) and have to reconstruct the intended architecture of it. They first have HUSACCT analyze the source code. Based on this analysis, HUSACCT builds an explorable hierarchical model which can be examined using various HUSACCT views. Figure 1 shows a module diagram of the implemented architecture (based on the high-level packages) and their dependencies. Each shown dependency is based on a summary of all concrete dependencies between both elements in the implemented system. The number of concrete dependencies is shown in the diagram and the dependencies with the highest numbers of concrete dependencies are additionally emphasized by representing them with thicker lines. After selection of a dependency arrow in the diagram, a tabular and sortable overview is given of all concrete dependencies. This overview contains information on the from- and to-artifacts (on class level), the line number in the source code file and the type of dependency. Clicking on a concrete dependency opens a code browser which highlights the line of code causing this dependency.

Using these views, the students have to examine the implemented architecture and reason about a probable intended architecture. They also have to examine the dependencies in detail, hereby being exposed to a variety of dependency types. Based on their findings, the students finally are asked to identify which elements which parts of the intended architecture represent. All results of this assignment had to be handed in by the students and were discussed in the following class meeting.

3.2 Assignment 2: Architecture Compliance Checking

In this assignment, the students are given the intended architecture of HUSACCT v1.0. A component model was provided (see Figure 2), which also formed the starting point of the HUSACCT implementation. All components had a clear responsibility, such as the analysis of the source code (Analyse component) or the definition

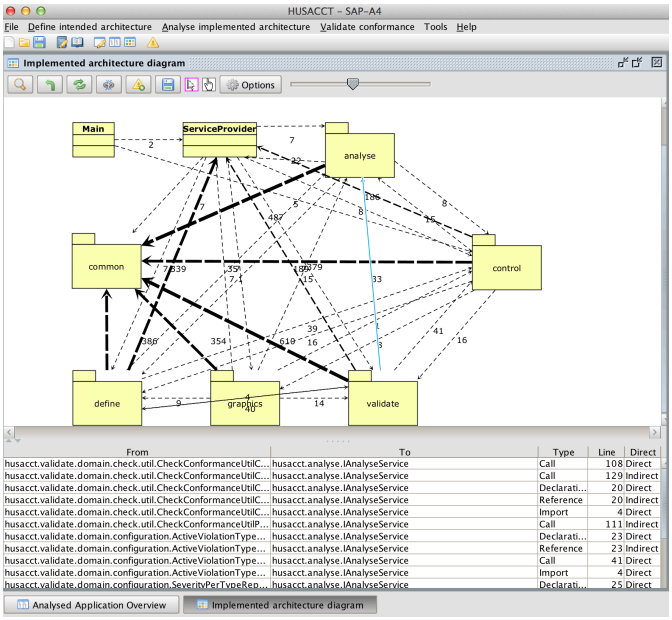

Figure 1: Visualization of implemented architecture and a tabular dependency overview

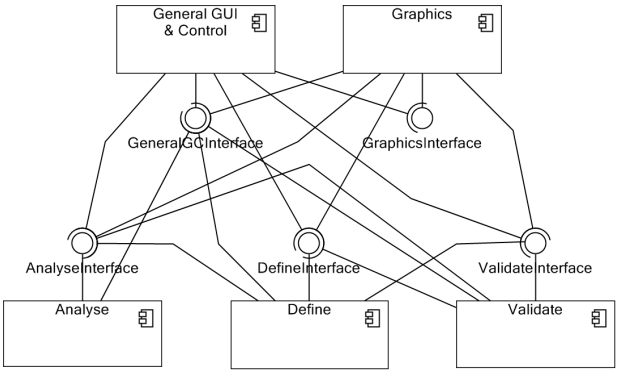

Figure 2: Component Model of HUSACCT v1.0

From-Module	Constraint	To-Module
Analyse	is not allowed to use	Define
Analyse	is not allowed to use	Validate
General GUI & Control	Is the only module allowed to use	Graphics

Figure 3: Example rules of intended architecture HUSACCT v1.0

of the intended architecture (Define component). Additionally, the rules that are applicable for the relations between the modules of the intended architecture and for the modules themselves (such as naming constraints) were given (see Figure 3 for an example). Finally, an overview of the mappings from the elements of the intended architecture to the software units in the analyzed source code (Figure 4) was provided. This information was used by the students to define the intended architecture of HUSACCT in the tool itself.

The students then had to execute the compliance check, examine the results, and to answer a series of questions such as "Which rule is most often violated and is this a problem?" or "Will it be easy to replace the existing user interface of Define by another

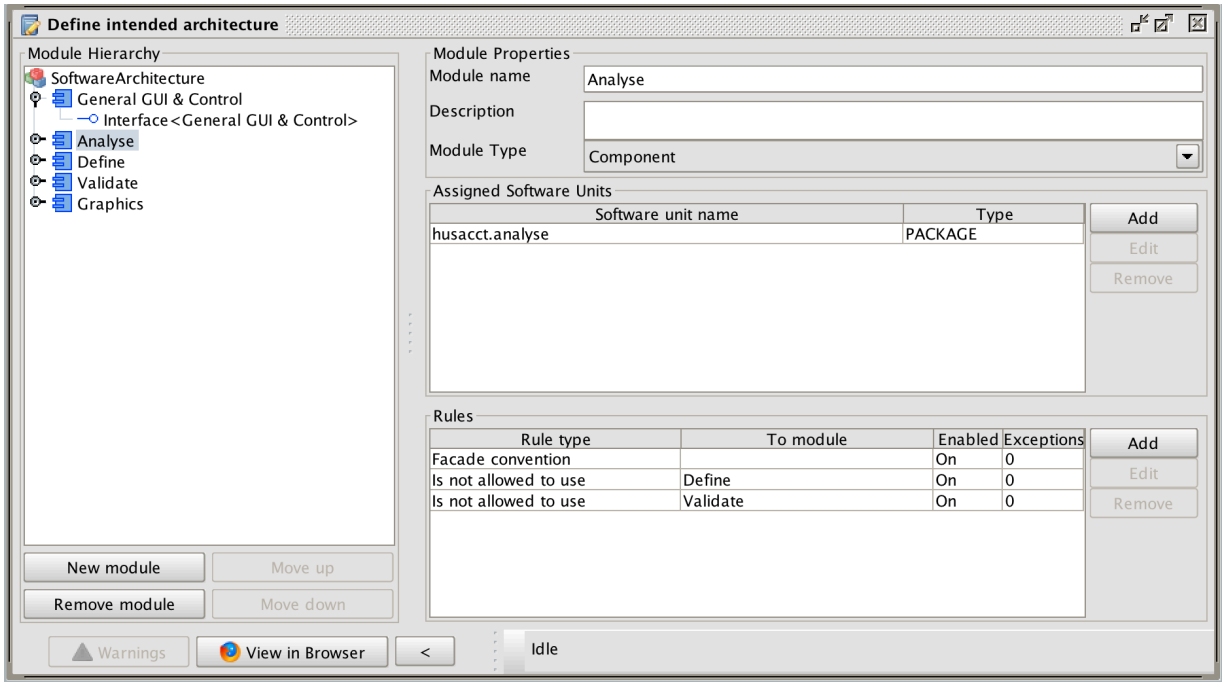

Figure 4: Define intended architecture view in HUSACCT, module hierarchy, mapping to software units and rule specification shown

implementation? Why?". As with the previous assignments, the students had to hand in their results which then were used for an extensive discussion in the next class meeting.

3.3 Assignment 3: Combination of reconstruction and compliance checking

This assignment is given only at the 3rd year students of HU University of Applied Sciences as part of the course *Advanced Software Architecture*. It is intended for applying both architecture reconstruction and compliance checking on a larger and more complex system and in a group of 3-4 students. The collaborative aspect encourages discussions between the students and increases the variety of e.g. potential solutions.

The students are first asked to analyze a larger open source system (up to 300.000 lines of code, a couple of examples were provided but the students were free to choose). They then have to determine a possible intended architecture through examining the analyzed modular elements and reasoning about possible design decisions made by the developers of the systems. Based on this likely intended architecture, the students had to identify possible applicable rules, e.g. rules of layering if layers were identified or communication via interfaces only if components were identified. Based on the results of the following compliance check, the students have to give advice on possible improvements for the system on both architectural and implementation level, making again the connection between these two more tangible.

4. Educational Value of HUSACCT

In our opinion, using HUSACCT in software architecture education is valuable, as it:

- supports the teaching of the procedural part of architecture reconstruction and architecture compliance checking;
- makes the connection between the intended architecture and the realized source code explicit and visible through the required mapping of architectural elements on software units (as shown in Figure 4);
- supports better understanding of semantically rich modular architectures [5], as there are two different representations of the intended architecture: as hierarchical tree and as diagram (see Figures 1 and 4);
- supports better understanding of the different types of rules that are related to the dependencies between different modules, as these rules need to be explicitly defined in HUSACCT (see Figure 4);
- shows the relation between a logical division into components in the architecture and the equivalents in the realized software system, as in HUSACCT all elements of the compliance checking process map to components in the software architecture (in Figure 4, the component Analyse is mapped to the package husacct.analyse);
- gives good insight in the variety of possible dependencies in source code, as all dependency types described in [4] are included in the source code analysis of HUSACCT and provided in the violation view (see Figure 5), including direct linking to the violation-causing source code;
- shows the direct relation between an architectural violation and the part of the source code that is responsible for it, again explicitly linking these high- and low-level aspects, hereby making the relevance of the architectural concepts more experienceable; and
- combines conceptual and procedural knowledge, two important knowledge dimensions as described in the revised Bloom's taxonomy [1].

However, we cannot provide empirical proof of these assumptions yet. In the future, we will validate the applicability of the tool in a large scale empirical study across different universities.

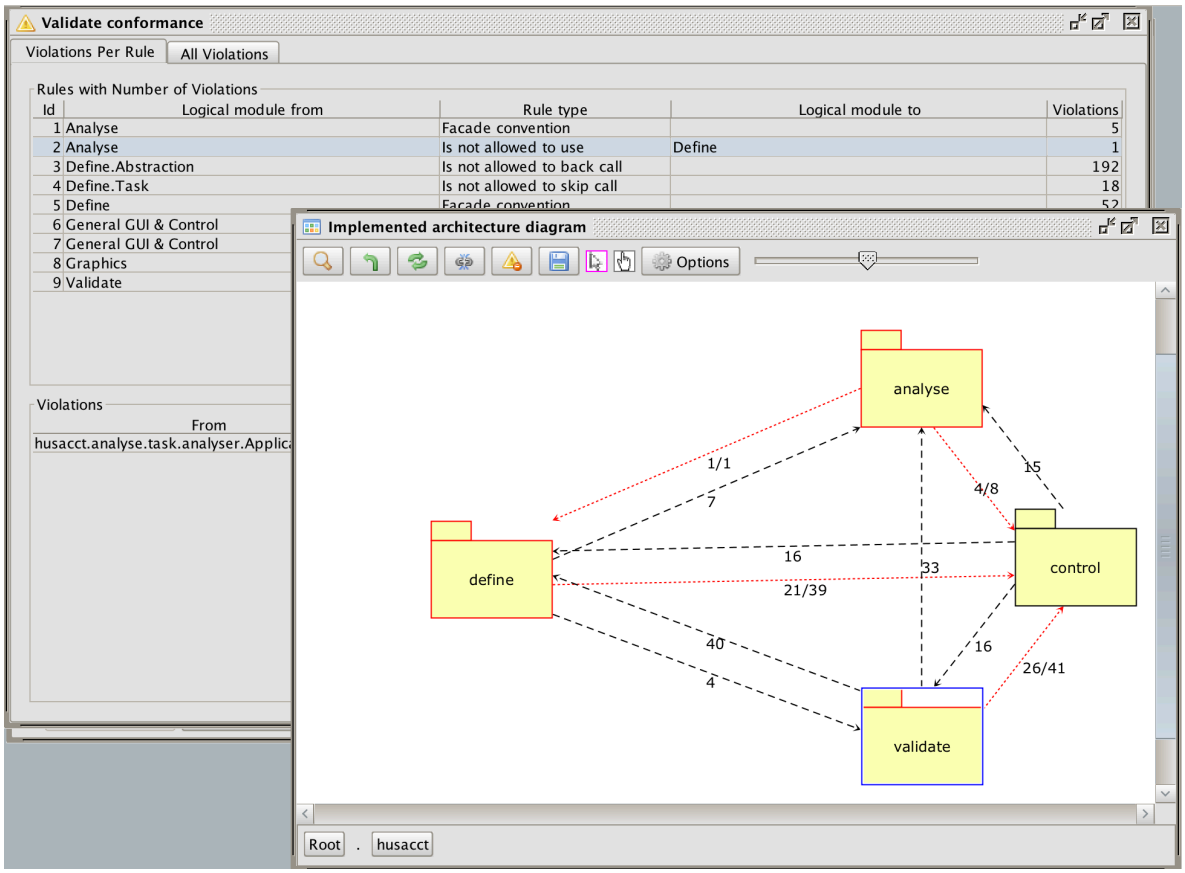

Figure 5: Violation report and implemented architecture view in HUSACCT (only parts are shown)

References

[1] L. W. Anderson and D. R. Krathwohl. *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Addison Wesley Longman, Inc, New York, 2001. ISBN 0321084055. .

[2] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*, volume 3rd editio. Addison-Wesley Professional, Oct. 2012. ISBN 0321815734, 9780321815736.

[3] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, and J. Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, 2010.

[4] L. Pruijt, C. Köppe, and S. Brinkkemper. On the Accuracy of Architecture Compliance Checking Support: Accuracy of Dependency Analysis and Violation Reporting. In *Proceedings of the International Conference on Program Comprehension, ICPC'13*, pages 172–181, San Francisco, CA, USA, 2013. ISBN 9781467330923.

[5] L. Pruijt, C. Köppe, and S. Brinkkemper. Architecture Compliance Checking of Semantically Rich Modular Architectures: A Comparison of Tool Support. In *Proceedings of the 29th International Conference on Software Maintenance, ICSM'13*, pages 220–229. IEEE Computer Society Press, 2013. .

[6] L. Pruijt, C. Köppe, J. van der Werf, and S. Brinkkemper. HUSACCT: Architecture Compliance Checking with Rich Sets of Module and Rule Types. In *Proceedings of the 29th IEEE/ACM International Conference on Automated Software Engineering (ASE 2014)*, pages 1–4, 2014. ISBN 9781450330138. . URL `http://dl.acm.org/citation.cfm?id=2642937.2648624`.